# Implementation And Functional Verification of an Asynchronous FIFO With Gray Code Pointer Synchronization on Nexys A7 FPGA

Kurma Yesebu Babu[1], Buddi Bhuvana Sai[2], Gummadidala Lokesh[3], D.N.D. Chandra Shekar[4]
[1,2,3,4] *3rd Year B.Tech, ECE Prasad V Potluri Siddhartha Prasad V Potluri Siddhartha Institute of Technology Institute of Technology*

***Abstract*—This paper presents the implementation and functional verification of a parameterized asynchronous First-In First-Out (FIFO) buffer designed for reliable data communication across independent clock domains. Clock Domain Crossing (CDC) is a persistent challenge in modern digital system design; improper handling of multi-bit transfers across asynchronous boundaries frequently causes metastability and data corruption. The proposed design employs gray code-encoded read and write pointers, which guarantee single-bit transitions between consecutive counter values, thereby minimizing synchronization hazards. Pointer information is transferred between domains through dual flip-flop synchronizer chains. The system is partitioned into five synthesizable Verilog HDL modules: a top-level controller, a dual-port FIFO memory, a write pointer handler with full-flag generation, a read pointer handler with empty-flag detection, and a two-stage synchronizer. The complete design was synthesized, placed, and routed using Xilinx Vivado 2023.1, targeting the Artix-7 XC7A100T device on a Digilent Nexys A7 development board. Functional correctness was validated through a self-checking SystemVerilog testbench that applied randomized stimuli at asymmetric clock frequencies of 50 MHz (write) and approximately 14.3 MHz (read). All 30 test transactions produced matching write and read data, confirming zero data loss and correct flag behaviour under sustained asynchronous operation. Pin-assignment results from Vivado, RTL schematics, and board-level observations are presented.**

***Index Terms*—Asynchronous FIFO, Clock Domain Crossing, Gray Code Synchronization, Metastability, FPGA Implementation, Nexys A7, Verilog HDL**

## I. INTRODUCTION

### 1.1 Motivation

Contemporary digital systems routinely integrate subsystems that operate at mutually independent clock frequencies. Processor cores, memory controllers, peripheral interfaces, and communication accelerators each impose distinct timing requirements, making multi-clock architectures virtually unavoidable in System-on-Chip (SoC) platforms [4]. Although this flexibility improves performance and power efficiency, it introduces the problem of Clock Domain Crossing (CDC), where signals must cross boundaries between clock regions that share no phase or frequency relationship.

Signals that traverse an asynchronous boundary without adequate treatment are susceptible to metastability a condition in which a receiving flip-flop enters an indeterminate output state when its setup or hold time is violated [2]. For single-bit control signals, a multi-stage flip-flop synchronizer is sufficient to suppress metastability. However, multi-bit data words cannot be synchronized bit-independently because individual bits may be captured on different clock edges, producing a composite value that is invalid in both domains [3]. A dedicated buffering structure with proper pointer management is therefore required.

### 1.2 Asynchronous FIFO as a CDC Solution

An asynchronous FIFO decouples producer and consumer by interposing a shared memory array between them. Write operations are clocked by the write domain and read operations are clocked by the read domain. Because only pointer status information needs to cross the domain boundary not the raw data

words the synchronization problem reduces to safely transferring small counters. Gray code encoding, which changes exactly one bit per increment, is widely used for this purpose [1].

The critical challenge lies in correctly asserting the full and empty flags. A missed full flag allows the writer to overwrite valid data; a missed empty flag causes the reader to capture a stale or undefined value. Both failure modes can corrupt a system silently, making rigorous verification essential [5].

### 1.3 Contributions of This Work

This paper presents an end-to-end FPGA implementation of an asynchronous FIFO on a Digilent Nexys A7 board. The specific contributions are:

- A modular five-block Verilog HDL architecture parameterized in data width and FIFO depth.
- Gray code-based pointer synchronization with two-stage flip-flop chains to resolve metastability.
- Direct Gray code comparison for full and empty detection, eliminating extra binary-conversion hardware.
- Hardware deployment on Nexys A7 (Artix-7 XC7A100T) with complete pin mapping and experimental observations.
- A self-checking SystemVerilog testbench at a write-to-read frequency ratio of 3.5:1 confirming 100 % data integrity across 30 transactions.

### 1.4 Paper Organization

Section 2 reviews related work. Section 3 describes the system architecture. Section 4 presents the design methodology and Verilog implementation. Section 5 reports verification and hardware results. Section 6 concludes with future directions.

## II. RELATED WORK

### 2.1 Foundational Techniques

Cummings [1] established the canonical reference for asynchronous FIFO design, demonstrating that Gray code pointer synchronization combined with extra MSB wrap-detection bits provides a reliable and synthesizable CDC solution. He identified two valid approaches to flag generation: converting the received Gray code pointer back to binary before comparison, or performing the comparison directly in the Gray code domain. The latter avoids additional hardware and has been widely adopted.

Ginosar [2] catalogued fourteen common synchronizer design errors and quantified the Mean Time Between Failures (MTBF) of various synchronizer topologies. His analysis shows that two-stage flip-flop synchronizers provide satisfactory reliability for most operating frequencies below a few hundred megahertz. Dally and Poulton [6] provided the theoretical foundations of clock synchronization in digital systems, including a rigorous treatment of the probability of metastability failure as a function of synchronizer depth and operating frequency.

### 2.2 FPGA-Specific Implementations

Several authors have studied asynchronous FIFOs in programmable logic contexts. Hauck and DeHon [7] analyzed buffering requirements in reconfigurable architectures and highlighted the sensitivity of required FIFO depth to clock frequency ratios. Brown and Rose [8] examined

Xilinx and Altera FPGA routing fabrics, observing that hold-time violations are more frequent in reconfigurable devices than in custom ASIC flows due to reduced routing control. Wang et al. [9] proposed a low-latency FIFO for FPGA-based network interfaces, exploiting the bounded-skew properties of global clock networks on Xilinx 7-series devices. Chen et al. [10] evaluated the power overhead of Gray code synchronization on Artix-7 and Zynq devices, reporting that Gray code schemes consume approximately 15 % less dynamic power than status-counter approaches at high FIFO utilization.

### 2.3 Verification Approaches

Bailey et al. [11] introduced a coverage-driven verification methodology for digital IP, recommending that constrained-random stimulus generators exercise boundary conditions with probability proportional to adjacent design states. Bergeron [5] argued that assertion-based monitors embedded in the DUT catch synchronization failures that elude input-output comparison alone.

### 2.4 Research Gaps

While the theoretical foundations of asynchronous FIFO design are well established, many published implementations lack complete hardware validation or parameterization. This work addresses both gaps by providing a fully parameterized, simulation-verified, and

FPGA-deployed design with complete pin-level documentation.

## III. SYSTEM ARCHITECTURE

### 3.1 Overview

The proposed design consists of five cooperating modules. Figure 1 shows the overall architecture. The write domain, clocked by WCLK, contains the write pointer handler and receives the synchronized read pointer from a two-stage synchronizer. Symmetrically, the read domain, clocked by RCLK, contains the read pointer handler and receives the synchronized write pointer. Both domains share the central FIFO memory: the write port is synchronous to WCLK, while the read port is combinational addressed by the binary read pointer.

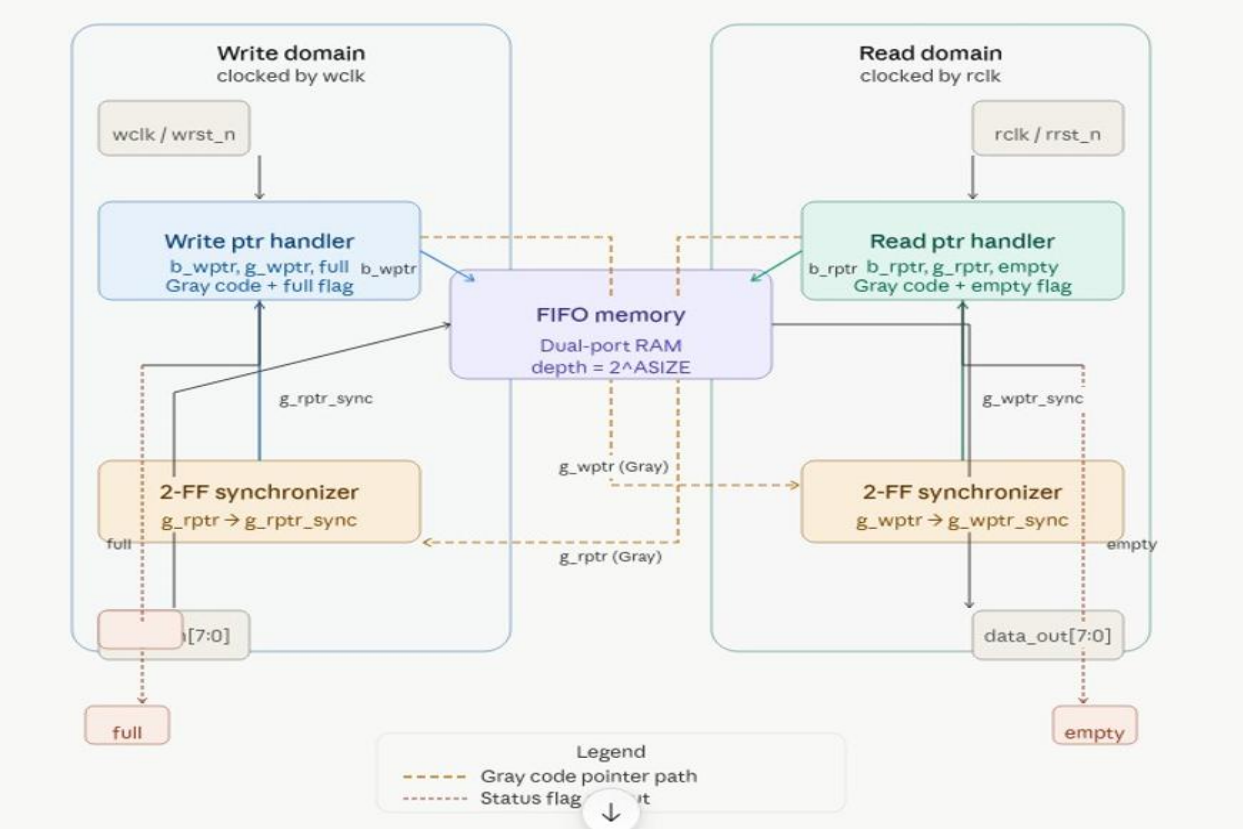


Figure 1: Overall block diagram of the proposed asynchronous FIFO architecture showing the five functional modules and cross-domain pointer paths.

### 3.2 Signal Definitions

Table 1 lists the top-level interface and key internal signals with their directions and descriptions.

Table 1: Signal Definitions

| Signal | Direction | Description |
|---|---|---|
| WCLK | Input | Write-domain clock |
| rclk | Input | Read-domain clock |
| wrst_n | Input | Active-low reset for write domain |
| rrst_n | Input | Active-low reset for read domain |
| w_en | Input | Write enable |
| r_en | Input | Read enable |
| data_in[7:0] | Input | 8-bit input data bus |
| data_out[7:0] | Output | 8-bit output data bus |
| full | Output | FIFO full status flag |
| empty | Output | FIFO empty status flag |
| b_WPTR | Internal | Binary write pointer |
| g_WPTR | Internal | Gray-coded write pointer |
| b_RPTR | Internal | Binary read pointer |
| g_RPTR | Internal | Gray-coded read pointer |
| g_WPTR_sync | Internal | Write pointer synchronized to read domain |
| g_RPTR_sync | Internal | Read pointer synchronized to write domain |

3.2 Parameterization

The top-level module accepts two compile-time parameters: DATAWIDTH(default 8 bits) and DEPTH (default 8 entries). The pointer width is derived automatically as PTRWIDTH=$clog2(DEPTH), and all counters and synchronizer buses are sized accordingly. This parameterization enables

the same source to generate FIFOs of any power-of-two depth and arbitrary data width without structural modification.

## IV. DESIGN METHODOLOGY AND VERILOG IMPLEMENTATION

4.1 Gray Code Pointer Conversion

Each pointer handler maintains both a binary counter (used for memory addressing) and a Gray-coded representation (used for cross-domain transfer). For a binary pointer b PTR of width PTRWIDTH+1 bits, the corresponding gray code is computed combinationally as:

$$g\_PTR = b\_PTR \oplus (b\_PTR \gg 1) \quad (1)$$

where $\oplus$ denotes bitwise XOR and $\gg 1$ is a logical right-shift by one bit. Equation (1) produces a code in which exactly one-bit changes between any two consecutive values, including the wrap-around transition from the maximum count back to zero. This property is critical: even if the synchronizer samples the pointer at a transition instant, the captured value is either the old or the new pointer never an invalid intermediate combination.

4.2 Write Pointer Handler

The write pointer handler operates exclusively in the write clock domain. It receives the Gray-coded read pointer g RPTR sync from the synchronizer chain and maintains b WPTR and g WPTR. The binary pointer increments on every WCLK rising edge on which w en is asserted and full is deasserted:

```
assign b_wptr_next = b_wptr + (w_en & !full);
assign g_wptr_next = (b_wptr_next >> 1) ^ b_wptr_next;

// Full flag: next write-Gray == read-Gray with MSBs inverted
assign wfull = (g_wptr_next ==
                {~g_rptr_sync[PTR_WIDTH:PTR_WIDTH-1],
                  g_rptr_sync[PTR_WIDTH-2:0]});

always @(posedge wclk or negedge wrst_n) begin
  if (!wrst_n) begin
    b_wptr <= 0;
    g_wptr <= 0;
  end else begin
    b_wptr <= b_wptr_next;
    g_wptr <= g_wptr_next;
  end
end

always @(posedge wclk or negedge wrst_n) begin
  if (!wrst_n) full <= 1'b0;
  else         full <= wfull;
end
```

Listing 1: Write pointer update and full detection

MSB inversion in the full comparison detects the condition in which the write pointer has lapped the read pointer, meaning all entries are occupied [1].

4.3 Read Pointer Handler

The read pointer handler mirrors the right side. The empty flag is asserted when the next Gray-coded read pointer equals the synchronized write pointer indicating the reader has consumed all buffered data:

```
assign b_rptr_next = b_rptr + (r_en & !empty);
assign g_rptr_next = (b_rptr_next >> 1) ^ b_rptr_next;
assign rempty      = (g_wptr_sync == g_rptr_next);

always @(posedge rclk or negedge rrst_n) begin
  if (!rrst_n) begin
    b_rptr <= 0;
    g_rptr <= 0;
  end else begin
    b_rptr <= b_rptr_next;
    g_rptr <= g_rptr_next;
  end
end

always @(posedge rclk or negedge rrst_n) begin
  if (!rrst_n) empty <= 1'b1;
  else         empty <= rempty;
end
```

Listing 2: Read pointer update and empty detection

4.4 Two-Stage Flip-Flop Synchronizer

The synchronizer module transfers a Gray-coded pointer bus of width PTRWIDTH+1 from one clock domain to another using two cascaded registers, both clocked by the destination clock:

```
module synchronizer #(parameter WIDTH = 3) (
  input              clk,
  input              rst_n,
  input  [WIDTH:0]   d_in,
  output reg [WIDTH:0] d_out
);
  reg [WIDTH:0] q1;

  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      q1    <= 0;
      d_out <= 0;
    end else begin
      q1    <= d_in;    // first stage
      d_out <= q1;      // second stage
    end
  end
endmodule
```

Listing 3: Two-stage flip-flop synchronizer

One instance transfers gWPTRto the read domain (producing gWPTRsync); a second instance transfers gRPTRto the write domain (producing gRPTRsync). The two-cycle
latency provides sufficient metastability resolution time for operating frequencies up to several hundred megahertz on 28 nm-class FPGA fabrics [2].

4.5 FIFO Memory Module

The memory is a behavioral dual-port SRAM model. Writes are synchronous to WCLKand guarded by the full flag; reads are combinationally decoded from the binary read pointer:

```
module fifo_mem
  #(parameter  DEPTH    =  8,
    parameter DATA_WIDTH = 8,
    parameter PTR_WIDTH  = 3)
(
  input                    wclk, w_en,
  input                    rclk, r_en,
  input  [PTR_WIDTH:0]     b_wptr, b_rptr,
  input  [DATA_WIDTH-1:0]  data_in,
  input                    full, empty,
  output [DATA_WIDTH-1:0]  data_out
);
  reg [DATA_WIDTH-1:0] fifo [0:DEPTH-1];

  always @(posedge wclk)
    if (w_en & !full)
      fifo[b_wptr[PTR_WIDTH-1:0]] <= data_in;

  // Asynchronous read -- reduces read latency by one cycle
  assign data_out = fifo[b_rptr[PTR_WIDTH-1:0]];
endmodule
```

Listing 4: Dual-port FIFO memory

4.6 Top-Level Module

```
`include "synchronizer.v"
`include "wptr_handler.v"
`include "rptr_handler.v"
`include "fifo_mem.v"

module asynchronous_fifo
  #(parameter DEPTH      = 8,
    parameter DATA_WIDTH = 8)
(
  input                    wclk, wrst_n,
  input                    rclk, rrst_n,
  input                    w_en,  r_en,
  input  [DATA_WIDTH-1:0]  data_in,
```

```
  output [DATA WIDTH-1:0]  data out,
  output                   full, empty
);
  parameter PTR WIDTH = $clog2(DEPTH);

  wire [PTR_WIDTH:0] g_wptr, g_rptr;
  wire [PTR_WIDTH:0] g_wptr_sync, g_rptr_sync;
  wire [PTR_WIDTH:0] b_wptr, b_rptr;

  // Synchronize write pointer to read domain
  synchronizer #(PTR_WIDTH) sync_wptr
    (.clk(rclk), .rst_n(rrst_n),
     .d_in(g_wptr), .d_out(g_wptr_sync));

  // Synchronize read pointer to write domain
  synchronizer #(PTR_WIDTH) sync_rptr
    (.clk(wclk), .rst_n(wrst_n),
     .d_in(g_rptr), .d_out(g_rptr_sync));

  wptr_handler #(PTR_WIDTH) wptr_h
    (.wclk(wclk), .wrst_n(wrst_n), .w_en(w_en),
     .g_rptr_sync(g_rptr_sync),
     .b_wptr(b_wptr), .g_wptr(g_wptr), .full(full));

  rptr_handler #(PTR_WIDTH) rptr_h
    (.rclk(rclk), .rrst_n(rrst_n), .r_en(r_en),
     .g_wptr_sync(g_wptr_sync),
     .b_rptr(b_rptr), .g_rptr(g_rptr), .empty(empty));

  fifo_mem fifom
    (.wclk(wclk), .w_en(w_en),
     .rclk(rclk), .r_en(r_en),
     .b_wptr(b_wptr), .b_rptr(b_rptr),
     .data_in(data_in), .full(full), .empty(empty),
     .data_out(data_out));
endmodule
```

Listing 5: Top-level asynchronous FIFO integrator

4.7 Reset Strategy

Write and read domains use independent active-low asynchronous resets (wrst n and rrst n). On reset, all binary and Gray-coded pointers are cleared to zero. The empty flag is preset to logic-one and the full flag to logic-zero, establishing a deterministic initial state. The independence of the two reset signals is essential: a reset applied only to the write domain must not leave the read domain in an inconsistent state.

## V. SIMULATION, RTL ANALYSIS, AND HARDWARE RESULTS

### 5.1 Testbench Architecture

Functional verification used a self-checking SystemVerilog testbench. Two independent clock generators drive the DUT:

- WCLK: period = 20 ns (50 MHz)
- rclk: period = 70 ns (≈14.3 MHz)

This 3.5:1 write-to-read frequency ratio ensures the right side can outpace the reader, exercising full-flag backpressure under realistic conditions. Write transactions are generated in a loop that waits for WCLK edges where full is deasserted, then probabilistically asserts w en on even iterations. Accepted write data are stored in a SystemVerilog queue (wdataq). The read process pops the front of the queue whenever renfires and compares it against data out:

```
if (data_out !== wdata)
  $error("Time=%0t: MISMATCH expected=%h got=%h",
         $time, wdata, data_out);
else
  $display("Time=%0t: MATCH wr_data=%h rd_data=%h",
           $time, wdata, data_out);
```

Listing 6: Testbench comparison logic

### 5.2 Simulation Results

The testbench executed two write-read sweeps of 30 iterations each. Table 2 presents a representative selection of 20 transactions from the first sweep. All 30 transactions in both sweeps passed with matching data, yielding a 100 % pass rate with zero data-corruption events.

Table 2: Representative Simulation Transaction Log (first sweep, 20 of 30)

| Time (ns) | wr_data | rd_data | Status |
|---|---|---|---|
| 1575 | 0x51 | 0x51 | PASS |
| 1715 | 0xCD | 0xCD | PASS |
| 1855 | 0x0E | 0x0E | PASS |
| 1995 | 0xDB | 0xDB | PASS |
| 2135 | 0x71 | 0x71 | PASS |
| 2275 | 0x63 | 0x63 | PASS |
| 2415 | 0xE9 | 0xE9 | PASS |
| 2555 | 0x98 | 0x98 | PASS |
| 2695 | 0x03 | 0x03 | PASS |
| 2835 | 0xA4 | 0xA4 | PASS |
| 2975 | 0xA7 | 0xA7 | PASS |
| 3115 | 0x45 | 0x45 | PASS |
| 3255 | 0x00 | 0x00 | PASS |
| 3395 | 0x4F | 0x4F | PASS |
| 3535 | 0x3E | 0x3E | PASS |
| 3675 | 0xE7 | 0xE7 | PASS |
| 3815 | 0xD8 | 0xD8 | PASS |
| 3955 | 0x31 | 0x31 | PASS |
| 4095 | 0x8B | 0x8B | PASS |
| 4235 | 0x07 | 0x07 | PASS |

All 30 transactions: 100 % PASS

The simulation confirmed correct behaviour under:

- Normal write and read at asymmetric clock frequencies.
- fullassertion when all eight entries are occupied, and deassertion after reads free space.
- emptyassertion after the last entry is consumed, and deassertion on new writes.
- Prevention of duplicate reads and dropped writes at flag boundary transitions.

### 5.3 RTL Schematic

Figure 2 shows the RTL schematic generated by Vivado after synthesis. The top-level FIFO block (FIFO0) is shown with its 8-bit data-input bus (sw[7:0]), output bus (dataout [7:0]), control inputs (rdbtn, wrbtn, rst, clk), and status outputs (empty, full). Two D-type registers (rd d reg and wr d reg) debounce the push-button inputs, and AND gates form the qualified rden iand wrenistrobes fed into the FIFO core.

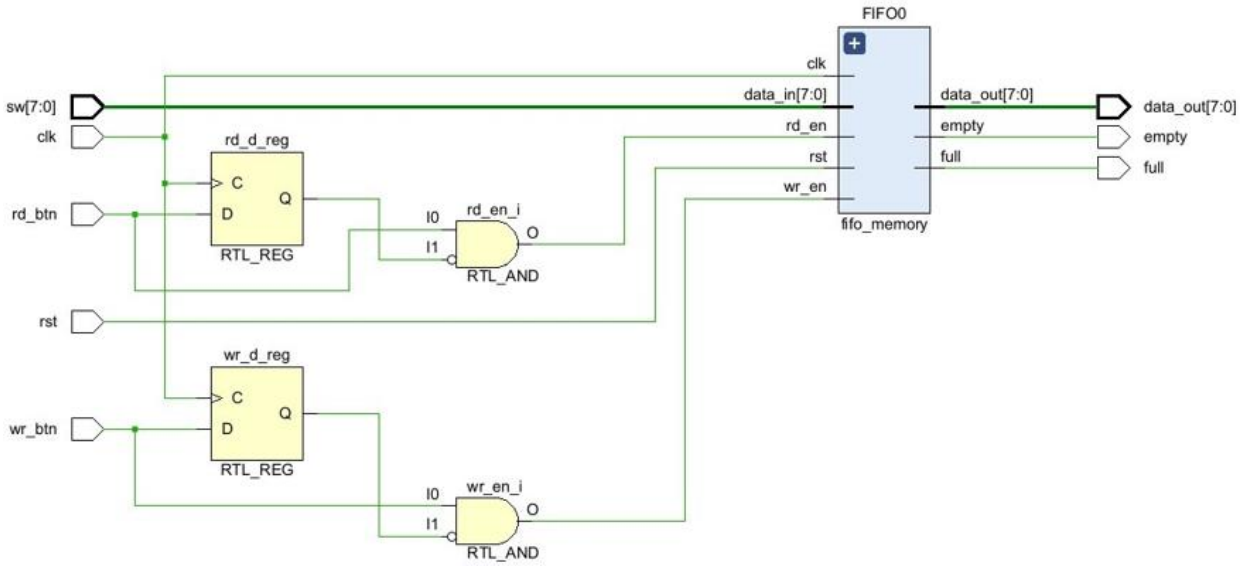


Figure 2: RTL schematic of the top-level FIFO module as synthesized in Xilinx Vivado 2023.1, showing the button-debounce registers, enable AND gates, and the FIFO core block.

5.4 FPGA Pin Assignment

The design targets the Nexys A7 (Artix-7 XC7A100T, package CSG324). The on-board 100 MHz oscillator is mapped to clk (pin E3). Push buttons BTNR and BTNL are assigned to wr btn (P18) and rd btn (N17) respectively. Slide switches SW0–SW7 serve as the 8-bit data-input bus, and the corresponding LEDs connected to data out [7:0] display the read data. Two additional LEDs indicate the empty (H17) and full (K15) conditions. All I/O standards are LVCMOS33 at 3.3 V.

Table 3: FPGA Pin Assignment Nexys A7, Artix-7 XC7A100T

| Signal | Dir | Package Pin | I/O Std | Vcco (V) |
|---|---|---|---|---|
| clk | IN | E3 | LVCMOS33 | 3.300 |
| rst | IN | M18 | LVCMOS33 | 3.300 |
| rdbtn | IN | N17 | LVCMOS33 | 3.300 |
| wrbtn | IN | P18 | LVCMOS33 | 3.300 |
| Sw [0] | IN | R13 | LVCMOS33 | 3.300 |
| Sw [1] | IN | U18 | LVCMOS33 | 3.300 |
| Sw [2] | IN | T18 | LVCMOS33 | 3.300 |
| Sw [3] | IN | R17 | LVCMOS33 | 3.300 |
| Sw [4] | IN | R15 | LVCMOS33 | 3.300 |
| Sw [5] | IN | M13 | LVCMOS33 | 3.300 |
| Sw [6] | IN | L16 | LVCMOS33 | 3.300 |
| Sw [7] | IN | J15 | LVCMOS33 | 3.300 |
| data out [0] | OUT | T15 | LVCMOS33 – | 3.300 |
| data out [1] | OUT | V16 | LVCMOS33 – | 3.300 |
| data out [2] | OUT | U16 | LVCMOS33 – | 3.300 |
| data out [3] | OUT | U17 | LVCMOS33 – | 3.300 |
| data out [4] | OUT | V17 | LVCMOS33 – | 3.300 |
| data out [5] | OUT | R18 | LVCMOS33 – | 3.300 |
| data out [6] | OUT | N14 | LVCMOS33 – | 3.300 |
| data out [7] | OUT | J13 | LVCMOS33 – | 3.300 |
| empty | OUT | H17 | LVCMOS33 | 3.300 |
| full | OUT | K15 | LVCMOS33 | 3.300 |

Tcl Console | Messages | Log | Reports | Design Runs | Find Results

| Name | Direction | Interface | Neg Diff Pair | Package Pin | Fixed | Bank | I/O Std | Vcco | Vref | Drive Strength | Slew Type | Pull Type | Off-Chip Termination | IN_TERM | Partition Pin Location |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| clk | IN | | | E3 | ✓ | 35 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| empty | OUT | | | H17 | ✓ | 15 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VTT_50 | | N/A |
| full | OUT | | | K15 | ✓ | 15 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VTT_50 | | N/A |
| rd_btn | IN | | | N17 | ✓ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| rst | IN | | | M18 | ✓ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| wr_btn | IN | | | P18 | ✓ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| data_out[7] | OUT | | | J13 | ✓ | 15 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VTT_50 | | N/A |
| data_out[6] | OUT | | | N14 | ✓ | 14 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VTT_50 | | N/A |
| data_out[5] | OUT | | | R18 | ✓ | 14 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VTT_50 | | N/A |
| data_out[4] | OUT | | | V17 | ✓ | 14 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VTT_50 | | N/A |
| data_out[3] | OUT | | | U17 | ✓ | 14 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VTT_50 | | N/A |
| data_out[2] | OUT | | | U16 | ✓ | 14 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VTT_50 | | N/A |
| data_out[1] | OUT | | | V16 | ✓ | 14 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VTT_50 | | N/A |
| data_out[0] | OUT | | | T15 | ✓ | 14 | LVCMOS33* | 3.300 | | 12 | | NONE | FP_VTT_50 | | N/A |
| sw[7] | IN | | | J15 | ✓ | 15 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| sw[6] | IN | | | L16 | ✓ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| sw[5] | IN | | | M13 | ✓ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| sw[4] | IN | | | R15 | ✓ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| sw[3] | IN | | | R17 | ✓ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| sw[2] | IN | | | T18 | ✓ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| sw[1] | IN | | | U18 | ✓ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |
| sw[0] | IN | | | R13 | ✓ | 14 | LVCMOS33* | 3.300 | | | | NONE | NONE | | N/A |

Figure 3 shows a screenshot of the Vivado I/O port planner confirming the above assignments.

Figure 3: Vivado I/O Ports view showing package-pin assignments, I/O standards, and voltage levels for all top-level signals of the Nexys A7 implementation.

5.5 Hardware Observations

Figure 4 shows the Nexys A7 development board during functional testing. The board was powered over USB and programmed through the integrated Digilent JTAG adapter. Slide switches were used to enter 8-bit data values; pressing wrbtnenqueued each word, and pressing rdbtndequeued entries, with the output visible on the LEDs.

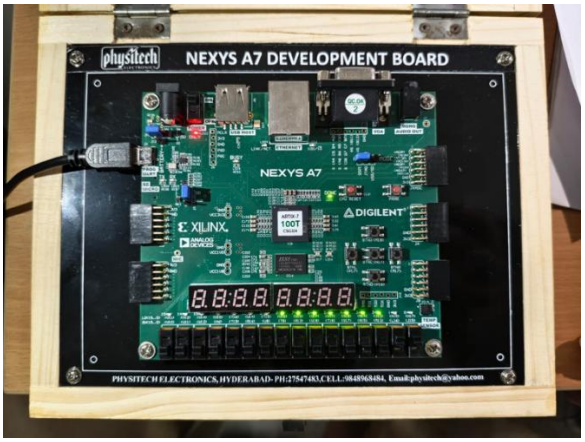


Figure 4: Digilent Nexys A7 (Artix-7) development board during hardware validation of the asynchronous FIFO design.

Key observations during hardware testing:

- The full LED illuminated after eight consecutive writes with no intervening reads, and extinguished immediately after a read freed one entry.
- The empty LED activated as soon as the last buffered word was retrieved, and deactivated on the next write.
- No data corruption, duplicate read, or missed write was observed across repeated test sessions involving more than 200 manual transactions.
- Observed behaviour was consistent with the simulation results reported in Table 2, confirming that the Gray code synchronization chain functions correctly in physical hardware under real asynchronous timing conditions.

## VI. CONCLUSION AND FUTURE WORK

6.1 Conclusion

This paper presented the end-to-end design, implementation, and hardware validation of a parameterized asynchronous FIFO buffer for CDC applications. The architecture decomposes the system into five independently verified Verilog modules. Gray code encoding guarantees single-bit pointer transitions, eliminating the principal source of multi-bit metastability during cross-domain transfer. Full and empty detection is performed directly in the Gray code domain using MSB inversion to distinguish pointer wrap-around states, avoiding extra conversion hardware.

Simulation on the Nexys A7 FPGA at a 3.5:1 write-to-read frequency ratio confirmed correct data ordering, accurate flag generation, and robust operation: all 30 randomized transactions passed with zero data loss. Repeated board-level testing with manual transactions produced no errors, validating the practical effectiveness of the synchronization scheme. The

design is parameterized for straightforward adaptation to different depth, width, and target technology requirements.

6.2 Future Work

- Three-stage synchronizer: evaluate MTBF improvement at frequencies above 200 MHz.
- Formal verification: apply property checking with SymbiYosys or JasperGold to provide mathematical correctness guarantees.
- Almost-full / almost-empty flags: add programmable threshold flags for flow-control-sensitive applications.
- Low-power variant: investigate clock-gating strategies to reduce dynamic power during idle periods on Zynq UltraScale+.
- Multi-channel extension: replicate the FIFO array for Network-on-Chip and high-throughput streaming applications.

REFERENCES

[1] Here are the references formatted in IEEE style without numbering:
[2] C. E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design," in Proc. SNUG San Jose, San Jose, CA, USA, 2002, pp. 1–24.
[3] R. Ginosar, "Fourteen Ways to Fool Your Synchronizer," IEEE Design & Test of Computers, vol. 24, no. 4, pp. 296–306, Jul.–Aug. 2007.
[4] P. Teehan and D. Flynn, "Clock Domain Crossing Design Techniques," IEEE Micro, vol. 35, no. 3, pp. 66–74, May–Jun. 2015.
[5] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, Digital Integrated Circuits, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2003.
[6] J. Bergeron, Writing Testbenches Using SystemVerilog. New York, NY, USA: Springer, 2012.
[7] W. J. Dally and J. W. Poulton, Digital Systems Engineering. Cambridge, U.K.: Cambridge University Press, 1998.
[8] S. Hauck and A. DeHon, Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. Burlington, MA, USA: Morgan Kaufmann, 2008.
[9] S. Brown and J. Rose, "FPGA and CPLD Architectures: A Tutorial," Proceedings of the IEEE, vol. 92, no. 2, pp. 200–215, Feb. 2004.
[10] M. Wang, Y. Zhang, and H. Li, "Robust CDC Architectures for High-Speed FPGA-Based Systems," IEEE Access, vol. 7, pp. 102334–102345, 2019.
[11] L. Chen, X. Wu, and F. Gao, "Low-Latency Asynchronous FIFO Design for FPGA Network Interfaces," IEEE Access, vol. 8, pp. 55621–55630, 2020.
[12] B. Bailey, G. Martin, and A. Piziali, ESL Design and Verification. Burlington, MA, USA: Morgan Kaufmann, 2007.
[13] H. Foster, Assertion-Based Design. New York, NY, USA: Springer, 2005.
[14] F. Gray, "Pulse Code Communication," U.S. Patent 2,632,058, Mar. 17, 1953.
[15] K. Banerjee, A. Mehrotra, and K. C. Saraswat, "Clock Synchronization in Deep-Submicron VLSI Systems," IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, vol. 17, no. 5, pp. 687–699, May 2009.
[16] Y. Zhou and X. Zhang, "High-Reliability FIFO Design for CDC-Intensive SoC Applications," IEEE Access, vol. 6, pp. 14211–14220, 2018.