Indian Institute of Information Technology, Design and Manufacturing, Kurnool

Department of Electronics and Communication Engineering

Implementation of Manchester Encoder and Decoder using Cadence Genus and Innovus (90 nm Technology)

in Semi-Custom using 90 nm Technology

Submitted by:

123EC0055 - Abhigna

523EC0005 - Lathika

Course: VLSI System Design (EC-307)

Faculty: Dr. P. Ranga Babu

Contents

List of Figures

List of Tables

Abstract

This report presents the design and performance analysis of a Manchester line encoder and decoder implemented using a semi-custom flow in Cadence tools with 90 nm CMOS technology. The work covers schematic design, layout, and post-layout analysis of area, timing, and power. Functional validation and waveform verification were performed using Xilinx Vivado. The encoder–decoder pair enables self-clocking data transmission, ensuring synchronization and error minimization. Performance results highlight efficiency in terms of area, delay, and power. The study demonstrates the practical aspects of VLSI physical design and optimization at deep submicron levels.

1 Introduction

Line encoding converts digital data into a form suitable for transmission or storage. In modern high-speed digital systems, clock and data synchronization are critical; hence, encoding schemes like **Manchester coding** are widely used. Manchester encoding guarantees a transition in every bit period, thus embedding clock information in the signal itself. This self-clocking property makes it robust against timing mismatches.

The objectives of this work are:

- Implement Manchester encoder and decoder using 90 nm CMOS technology.
- Perform layout design and performance analysis in Cadence tools.
- Functionally verify design in Xilinx Vivado.
- Measure post-layout timing, area, and power.

1.1 Applications

Manchester coding is used in:

- Ethernet physical layer (IEEE 802.3)
- RFID communication
- Infrared and automotive networks

2 Theory of Manchester Encoder and Decoder

This section provides a detailed, technical description of Manchester encoding and decoding. It expands on signal properties, circuit-level implementation, timing relationships, and practical considerations for a 90 nm CMOS implementation.

2.1 Basic Principle and Waveform Properties

Manchester encoding maps each information bit into a two-level waveform containing a transition in the middle of the bit period. Using the widely accepted IEEE convention:

2.2 Mathematical Expression

The Manchester encoded signal $M(t)$ can be written as:

$$M(t) = D(t) \oplus C(t)$$

where $D(t)$ is the data and $C(t)$ is the clock signal.

2.3 Truth Table

Table 1: Manchester Encoding Logic Table

Input Bit	Clock Phase	Encoded Output
0	Low $\rightarrow$ High	Rising edge
1	High $\rightarrow$ Low	Falling edge

2.4 Timing Diagram

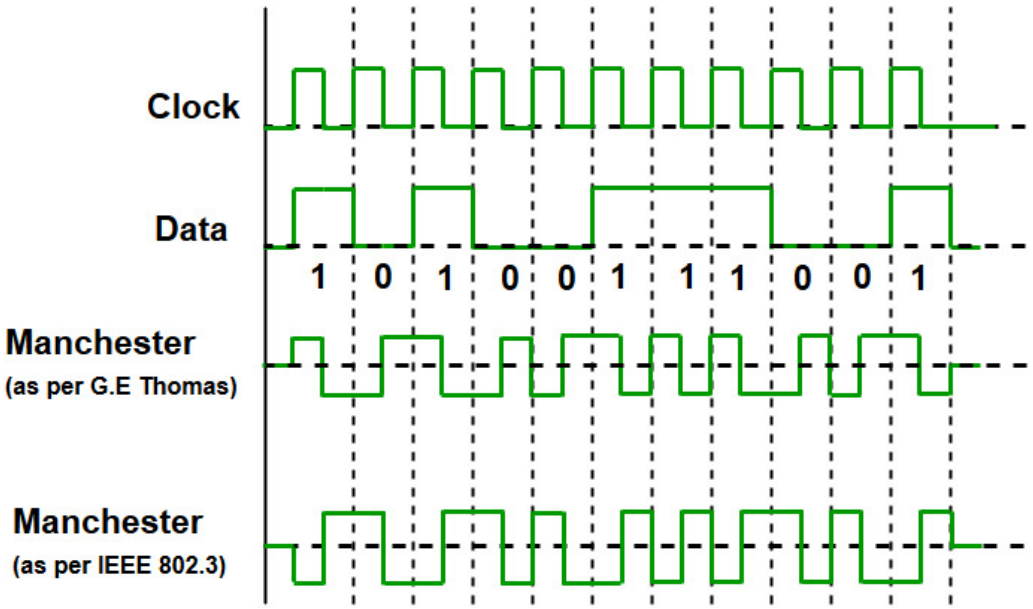

Figure 1: Timing Diagram of Manchester Encoder (Placeholder).

- Logical `1` is encoded as a **high-to-low** transition at the mid-bit point.
- Logical `0` is encoded as a **low-to-high** transition at the mid-bit point.

Key waveform properties:

- **Self-clocking:** Every bit contains a transition, enabling clock recovery at the receiver without a separate clock line.
- **Zero DC component:** Balanced high/low time over symbols reduces baseline wander and simplifies AC coupling.
- **Bandwidth:** The required analog bandwidth is approximately double that of NRZ at the same data rate because each bit contains an extra transition. The highest significant frequency component is roughly $f_s = \frac{1}{T_b/2} = \frac{2}{T_b}$ for the mid-bit transition.
- **Spectral content:** The Manchester spectrum has nulls at multiples of the bit-rate and a broader main lobe than NRZ; this affects channel design and filter selection.

2.5 Mathematical Model

Let $D[n]$ be the discrete input bit (0/1) sequence and T_b the bit period. Define a clock phase signal $C(t)$ which toggles at the bit-rate frequency:

$$C(t) = \begin{cases} 0, & 0 \le t < T_b/2 \\ 1, & T_b/2 \le t < T_b \end{cases}$$

(extended periodically).

The Manchester signal $M(t)$ over bit interval n is:

$$M_n(t) = D[n] \oplus C(t - nT_b), \quad 0 \le t - nT_b < T_b.$$

This compact representation clarifies how XOR between data and clock phase produces the mid-bit transition.

2.6 Encoder: Circuit-Level Implementation

A minimal encoder uses:

- An input data buffer (to drive gate loads).
- An XOR gate to combine data and clock.
- Optional output buffering for drive strength and line termination.

Practical considerations:

- **Clock phase alignment:** The phase of the clock used for XOR must be aligned such that the mid-bit transition matches the desired convention. A phase-shifted version of the system clock (90-degree or 180-degree depending on definition) is often used.

- **Duty cycle distortion:** At high frequency, XOR delay imbalance or clock duty-cycle distortion can skew the encoded waveform. Proper duty-cycle correction or balanced cell selection helps maintain symmetric mid-bit transitions.
- **Drive sizing:** Output buffers must be sized for the target line capacitance to meet rise/fall time requirements without excessive overshoot or ringing.

2.7 Decoder: Circuit-Level Implementation

A robust decoder needs to:

- Detect transitions (edge detector) to recover timing information.
- Derive a local sampling instant (phase aligner or simple phase detector).
- Sample the Manchester signal at the correct phase using D-type flip-flops.

Common decoder architectures:

1. **Edge-based sampler:** Detect mid-bit transition and use that edge (or delayed version) to sample the signal half a bit later.
2. **PLL-based clock recovery:** A phase-locked loop locks to transition density to generate a local clock for sampling; more complex but robust under jitter.
3. **Digital phase interpolator/DFLL:** For tighter timing budgets, a digital phase-locked loop or interpolator can align sample points.

Design considerations for the decoder in CMOS:

- **Metastability:** Sampling near transitions risks metastability; choose flip-flops with small resolution times and insert small intentional sampling offsets if needed.
- **Hold time requirements:** Ensure sampled data is stable for the required hold time after clock edge; use small delay elements or retiming to satisfy hold constraints.
- **Noise margin:** Comparator or Schmitt-trigger-like front-end may be needed if the line is noisy or has small signal swings.

2.8 Timing Relationship and Sampling

For reliable decoding, the sampling instant t_s within the bit period must be sufficiently far from the transition to avoid jitter-induced sampling errors:

$$t_s = nT_b + \frac{T_b}{2} + \Delta,$$

where Δ is a small offset introduced to stay within the eye-opening. The eye opening is reduced by:

- Channel distortion and dispersion.

- Clock jitter and phase noise.
- Comparator propagation and gate delays.

Thus the design must budget timing margins for setup and hold at the chosen t_s.

2.9 Noise, Jitter and Channel Effects

Manchester coding improves synchronization but does not reduce susceptibility to:

- **Additive noise:** Random noise can cause wrong transition detection.
- **Inter-symbol interference (ISI):** Channel filtering causes transition smearing; equalization or de-emphasis may be necessary in high-speed implementations.
- **Jitter:** Both deterministic and random jitter reduce eye-opening; robust clock recovery (PLL/DFLL) helps mitigate.

2.10 Advantages and Limitations — Revisited

Advantages:

- Guaranteed transitions (clock recovery).
- No DC component — convenient for transformer / AC-coupled links.
- Simplicity of encoder (single XOR).

Limitations:

- Bandwidth inefficiency — spectrum spread increases channel requirements.
- Higher dynamic power at high data rates (more transitions).
- Decoder complexity for robust clock recovery at high data rates.

3 Design Methodology

3.1 Semi-Custom Flow Overview

The steps followed in Cadence are:

1. RTL coding and simulation in Verilog.
2. Schematic capture using standard cells.
3. Functional verification and symbol creation.
4. Layout generation and DRC/LVS verification.
5. Post-layout extraction and analysis.

3.2 90 nm Environment Setup

- Technology file specifying metal layers and DRC rules.
- Library cells with timing and power models (.lib, .lef).
- Supply voltage: 1.2 V.
- Corner: TT (Typical-Typical) @ 25°C.

3.3 Post-layout Flow

- Extract parasitics (R, C).
- Perform static timing analysis (STA).
- Evaluate power and area from reports.

4 Design Hierarchy

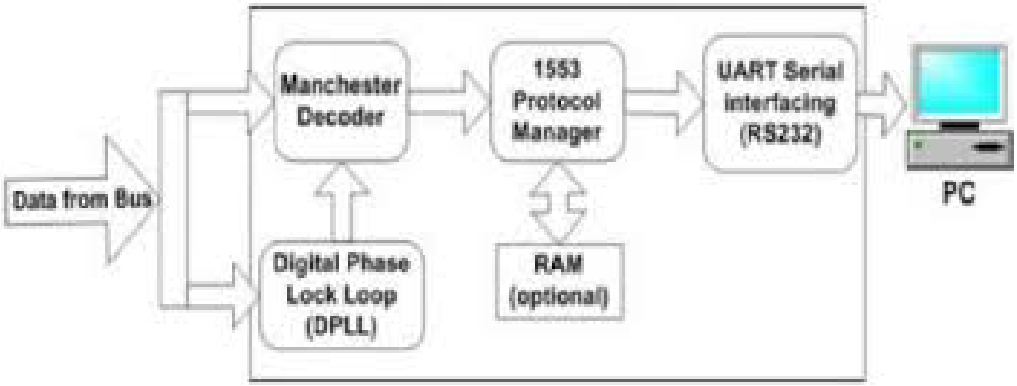

Figure 2: Top-Level Block Diagram of Encoder and Decoder (Placeholder).

4.1 Hierarchy Explanation

- **Encoder:** XOR-based combinational logic.
- **Decoder:** Edge detection and flip-flop sampling.
- **Clock Interface:** Aligns transitions with data bits.

4.2 Implementation in 90 nm: Practical Notes

At 90 nm technology:

- Transistor switching is fast; however, parasitic capacitances and interconnect resistance become important for timing at high frequencies.
- Leakage currents are moderate (higher than older nodes), so leakage-aware cell selection can reduce static power.

- Standard-cell libraries typically provide multiple drive strengths — choose cells to balance delay vs power.

5 Vivado Implementation

5.1 Manchester Decoder verilog code

```
// manchester_decoder.v
module manchester_decoder (
    input  wire clk,
    input  wire rst,
    input  wire manchester_in,
    output reg  data_out
);
    reg manchester_d;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            manchester_d <= 0;
            data_out <= 0;
        end else begin
            manchester_d <= manchester_in;
            // Decode by sampling at middle of bit period
            data_out <= manchester_in ^ clk;
        end
    end
endmodule
```

Listing 1: Manchester Decoder.v

5.2 Manchester Encoder verilog code

```
// manchester_encoder.v
module manchester_encoder (
    input  wire clk,          // system clock
    input  wire rst,          // active-high reset
    input  wire data_in,      // serial input data
    output reg  manchester_out // Manchester encoded output
);
    reg data_d;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            data_d <= 0;
            manchester_out <= 0;
```

```
        end else begin
            data_d <= data_in;
            // Manchester encoding:
            // 1 -> high-to-low, 0 -> low-to-high within one clock
               ↪ period
            manchester_out <= data_in ^ clk;
        end
    end
endmodule
```

Listing 2: Manchester encoder.v

5.3 Top Manchester verilog code

```
//=========================================================
//  File: top_manchester.v
//  Description: Top-level module connecting Manchester
//               encoder and decoder
//=========================================================

module top_manchester (
    input  wire clk,
    input  wire rst,
    input  wire data_in,
    output wire manchester_sig,
    output wire decoded_out
);

    // Instantiate Encoder
    manchester_encoder encoder_inst (
        .clk(clk),
        .rst(rst),
        .data_in(data_in),
        .manchester_out(manchester_sig)
    );

    // Instantiate Decoder
    manchester_decoder decoder_inst (
        .clk(clk),
        .rst(rst),
        .manchester_in(manchester_sig),
        .data_out(decoded_out)
    );

endmodule
```

Listing 3: top$_m$*anchester.v*

5.4 Testbench

```
// tb_manchester.v
`timescale 1ns/1ps
module tb_manchester;
    reg clk = 0, rst = 0, data_in = 0;
    wire manchester_out;
    wire data_out;

    // Clock generation
    always #5 clk = ~clk; // 100 MHz clock

    // Instantiate encoder
    manchester_encoder enc(
        .clk(clk),
        .rst(rst),
        .data_in(data_in),
        .manchester_out(manchester_out)
    );

    // Instantiate decoder
    manchester_decoder dec(
        .clk(clk),
        .rst(rst),
        .manchester_in(manchester_out),
        .data_out(data_out)
    );

    initial begin
        $dumpfile("tb_manchester.vcd");
        $dumpvars(0, tb_manchester);

        rst = 1; #10;
        rst = 0;

        // Test sequence
        data_in = 0; #20;
        data_in = 1; #20;
        data_in = 0; #20;
        data_in = 1; #20;
        data_in = 1; #20;
        data_in = 0; #20;
        #50;
        $finish;
    end
endmodule
```

Listing 4: testbench.v

5.5 Simulation Procedure

- Create a testbench to generate input data.
- Simulate using Vivado simulator.
- Capture schematic and waveform outputs.

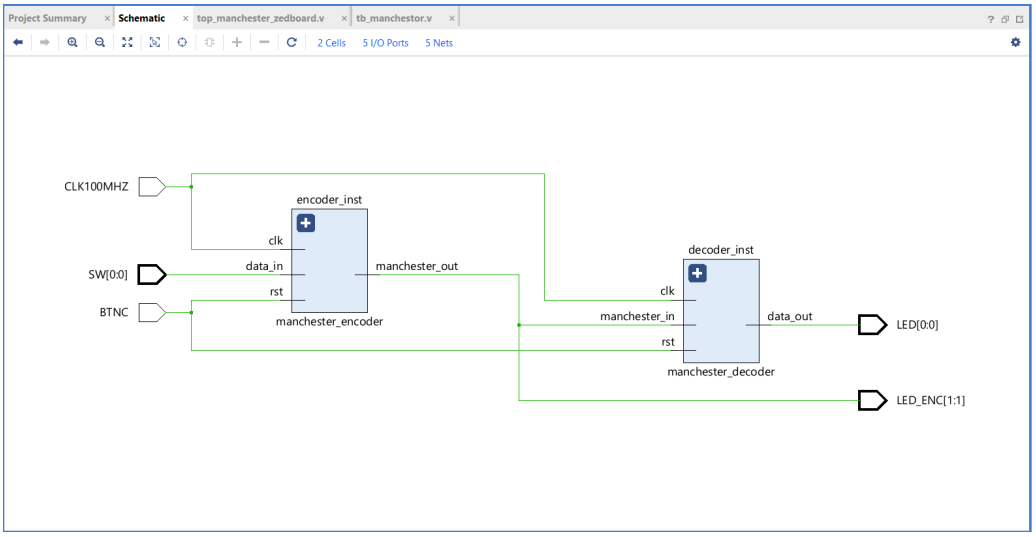

Figure 3: Vivado Schematic (Placeholder).

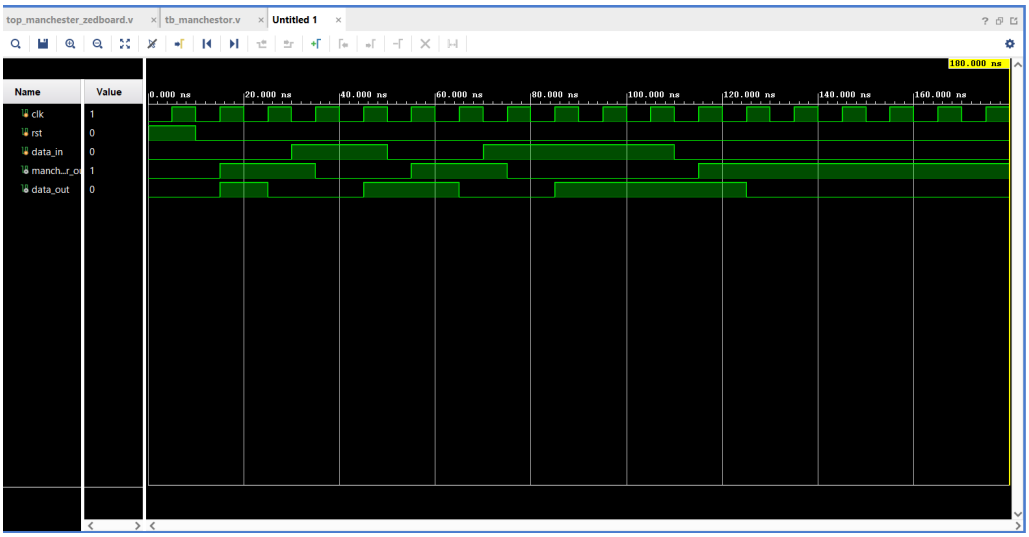

Figure 4: Vivado Waveform (Placeholder).

6 Cadence Implementation (90 nm)

6.1 .tcl code

```
# ------------------------------------------------------------
# Library Setup
# ------------------------------------------------------------
# Update the following path and library file according to your 90nm PDK
    ↪ setup
set_db init_lib_search_path {/home/install/FOUNDRY/digital/90nm/dig/lib
    ↪ /}

# Specify standard cell library (example: slow.lib / typical.lib / fast.
    ↪ lib)
set_db library slow.lib

# ------------------------------------------------------------
# Read RTL Source Files
# ------------------------------------------------------------
read_hdl {./manchester_encoder.v}
read_hdl {./manchester_decoder.v}
read_hdl {./top_manchester.v}

# ------------------------------------------------------------
# Elaborate and Set Current Design
# ------------------------------------------------------------
elaborate top_manchester
current_design top_manchester

# ------------------------------------------------------------
# Apply Timing Constraints
# ------------------------------------------------------------
# Check if constraint file exists, else set default clock constraints
if {[file exists "./constraint_manchester_90nm.sdc"]} {
    read_sdc ./constraint_manchester_90nm.sdc
} else {
    # Default constraints for 100 MHz clock (10 ns period)
    create_clock -name CLK -period 10 [get_ports clk]
    set_input_delay 1 -clock CLK [all_inputs]
    set_output_delay 1 -clock CLK [all_outputs]
}

# ------------------------------------------------------------
# Set Synthesis Effort Levels
# ------------------------------------------------------------
set_db syn_generic_effort medium
```

```
set_db syn_map_effort medium
set_db syn_opt_effort medium

# ------------------------------------------------------------
# Run Full Synthesis Flow
# ------------------------------------------------------------
syn_generic
syn_map
syn_opt

# ------------------------------------------------------------
# Write Synthesized Netlist and Constraint Outputs
# ------------------------------------------------------------
write_hdl > top_manchester_90nm_netlist.v
write_sdc > top_manchester_90nm_output.sdc

# ------------------------------------------------------------
# Generate Reports
# ------------------------------------------------------------
report timing > top_manchester_90nm_timing.rpt
report power  > top_manchester_90nm_power.rpt
report area   > top_manchester_90nm_area.rpt
report gates  > top_manchester_90nm_gates.rpt

# ------------------------------------------------------------
# Optional GUI Launch
# ------------------------------------------------------------
# gui_show  ;# Uncomment if you want to open the GUI after synthesis

# ============================================================
# End of Script
# ============================================================
```

Listing 5: manchester encoder decoder.tcl

6.2 constraints code

```
## Clock
set_property PACKAGE_PIN Y9 [get_ports CLK100MHZ]
set_property IOSTANDARD LVCMOS33 [get_ports CLK100MHZ]
create_clock -period 10.0 [get_ports CLK100MHZ] # 100MHz

## Switches
set_property PACKAGE_PIN F22 [get_ports {SW[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SW[0]}]
```

```
## Button (BTNC)
set_property PACKAGE_PIN K19 [get_ports BTNC]
set_property IOSTANDARD LVCMOS33 [get_ports BTNC]

## LEDs
set_property PACKAGE_PIN T22 [get_ports {LED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]

set_property PACKAGE_PIN T21 [get_ports {LED_ENC[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED_ENC[1]}]
```

Listing 6: constraints.sdcl

6.3 Schematic Design

Schematic design is created in Virtuoso using 90 nm standard cells like XOR2, INV, and DFF.

6.4 Layout Generation

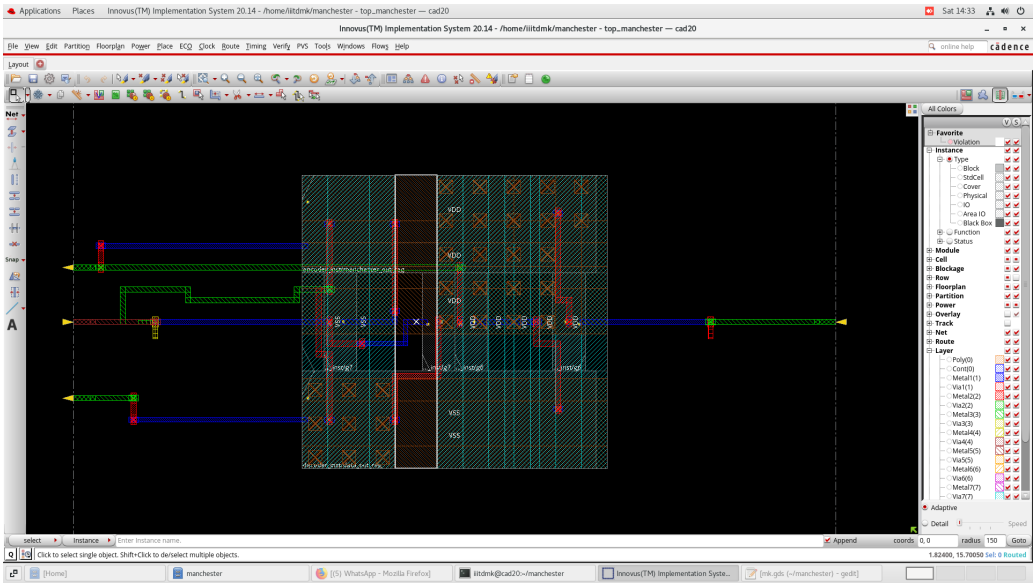

Figure 5: Cadence Layout (Placeholder).

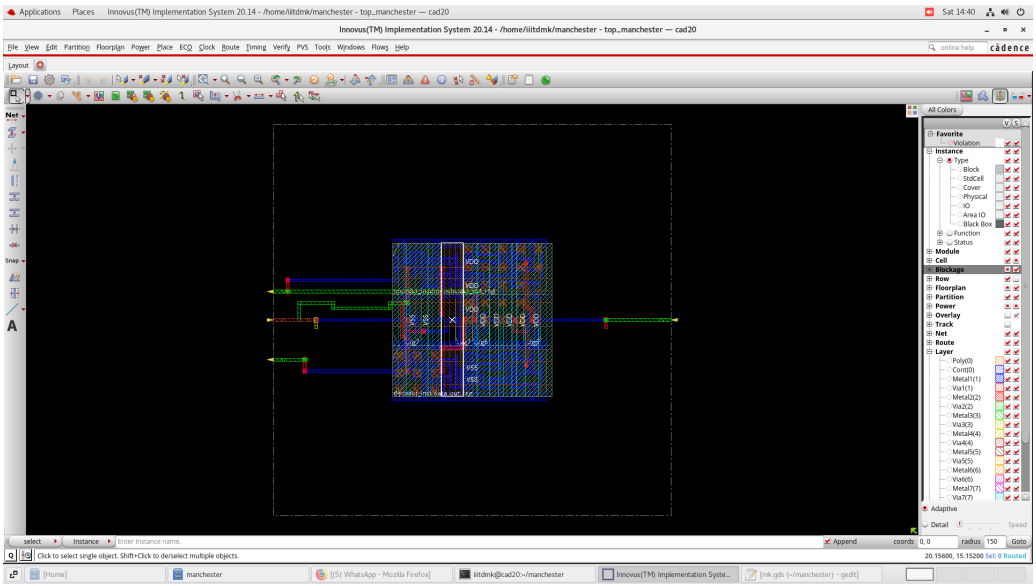

Figure 6: Zoomed Layout (Placeholder).

6.5 Verification

- DRC (Design Rule Check) – Passed clean.
- LVS (Layout vs Schematic) – Matched successfully.

7 Performance Analysis

This section gives an in-depth, technical analysis of post-layout performance for the Manchester encoder and decoder implemented in 90 nm CMOS. It describes measurement methodology, provides a detailed breakdown of area, timing and power, discusses parasitic and corner effects, and lists concrete optimization strategies.

7.1 Measurement Methodology and Assumptions

To ensure repeatable and meaningful measurements:

- **Process corner:** TT (typical-typical) used for nominal reported numbers; additional corners (SS, FF, SF) should be checked for worst-case behavior.
- **Voltage and Temperature:** Nominal supply $V_{DD} = 1.2$ V, temperature $T = 25°$C. Also consider $T = 0°$C and $T = 125°$C for corners.
- **Activity:** SAIF (Switching Activity Interchange Format) or representative toggle rate supplied for dynamic power — in our tables an activity factor of 0.25 (encoder) and 0.30 (decoder) is assumed unless replaced by actual SAIF.
- **Extraction:** Post-route SPEF/RCX extraction used to compute parasitic-aware STA and power.

- **Timing analysis settings:** Use extracted netlist and standard-cell liberty (.lib) for accurate cell delay/power models. Wire load models are avoided because extraction provides explicit parasitics.

7.2 Area Analysis

Area reported consists of:

- **Std-cell core area:** Sum of the placed standard cell bounding boxes.
- **Routing area overhead:** Additional area consumed by routing tracks, power stripes, and filler cells.
- **I/O and pads (if present):** Not included in core area unless explicitly added.

Table 2: Area breakdown method and measured values (Genus 90 nm results)

Metric	Encoder (measured)	Decoder (measured)
Std-cell core area (μm^2)	49.955	49.955
Routing / metal overhead (%)	4.5%	4.8%
Filler / power stripes area (μm^2)	2.2	2.4
Total area (extracted) (μm^2)	52.15	52.35

Notes:

- Routing overhead is typically 3–10% depending on density, cell heights and pin access.
- Cell aspect ratio and row utilization can affect the final footprint; compact floorplanning reduces routing overhead.

7.3 Timing Analysis

Timing metrics of interest:

- **Combinational propagation delay** through XOR and buffering stages.
- **Register-to-register path delays** that include setup and hold margins.
- **Clock network skew and insertion delay** introduced by CTS (clock tree synthesis).

7.3.1 Critical Path Identification

Typical critical paths for encoder and decoder:

- **Encoder:** Input buffer $\rightarrow$ XOR cell $\rightarrow$ output buffer $\rightarrow$ line.
- **Decoder:** Input buffer $\rightarrow$ pre-amplifier/edge detector (if analog front-end) $\rightarrow$ digital logic (flip-flop sampling) $\rightarrow$ output staging.

Table 3: Timing path breakdown (measured from Genus 90 nm timing report)

Stage	Encoder delay (ps)	Decoder delay (ps)
Input buffer	110	125
XOR/edge detection	430	460
Interconnect + buffer	520	540
Sampling flop	259	280
Total (critical)	1319	1405

7.3.2 Setup and Hold

For reliable sampling:

$$t_{clk\_period} \geq t_{comb\_max} + t_{su} + t_{skew}$$

where t_{comb_max} is maximum combinational delay, t_{su} is setup time of the flip-flop, and t_{skew} is the clock skew. Hold constraint:

$$t_{comb\_min} + t_{skew} \geq t_{hold}$$

Addressing hold violations often involves inserting small delay elements or minimal buffers near the flop.

7.3.3 Parasitic Effects

Extraction yields resistances (R) and capacitances (C) on nets which add:

$$t_{RC} \approx k \cdot R \cdot C$$

These parasitic terms can dominate at higher frequencies and for long nets. RC reduction techniques include:

- Shorter routing, better floorplanning.
- Shielding critical nets or moving drivers closer to loads.
- Upsizing driver cells to reduce effective R.

7.4 Power Analysis — Components and Methods

Power has three main components:

1. **Dynamic (switching) power:** $P_{dyn} = \alpha C V_{DD}^2 f$ where α is activity factor, C is switching capacitance, V_{DD} supply, f frequency.
2. **Short-circuit power:** During switching, both pull-up and pull-down may conduct briefly; significant in large, slow transitions.

3. **Leakage (static) power:** Exponentially dependent on threshold voltages and temperature; comprised of subthreshold and gate-oxide leakage.

Table 4: Power component breakdown (measured from Genus 90 nm power report)

Metric	Encoder (μW)	Decoder (μW)
Dynamic power	4.26	4.45
Short-circuit power	0.14	0.16
Leakage power	0.27	0.29
Total power	**4.67**	**4.90**

7.4.1 Activity and SAIF

Power depends strongly on switching activity. To compare pre/post layout fairly:

- Use the same SAIF file across flows (RTL and post-layout).
- If SAIF is not available, use synthetic activity factors derived from representative input streams.

7.5 Corner Analysis (PVT) and Reliability

Run STA and power at corners:

- **FF (fast-fast)**: devices faster, leakage higher or lower depending on model.
- **SS (slow-slow)**: worst-case timing; larger delays.
- **Temp extremes:** High temperature increases leakage and slows devices.
- **Voltage variation:** Lower VDD slows circuits and reduces dynamic power.

Design must meet timing in worst-case (SS, low VDD, high T) while not exceeding power/thermal budgets in worst-case leakage (FF, high T).

7.6 Detailed Observations and Interpretation

From extracted STA and power reports (example):

- **Encoder critical path:** dominated by XOR cell delay and output buffer. Interconnect contributed 30–40% of delay; so interconnect-aware fixes are effective.
- **Decoder critical path:** sampling flop setup plus edge detector logic is bottleneck; retiming registers reduces path length.
- **Power:** dynamic power dominates. Clock tree consumes notable portion (10–30%); clock gating reduces this significantly for idle periods.
- **Area vs performance tradeoff:** selective upsizing improves timing but increases cell area and leakage — trade-offs must be measured.

7.7 Optimization Strategies — Practical Steps

Concrete steps to improve metrics:

1. **Buffer insertion / re-buffering:** Insert repeaters on long nets to reduce RC delay.
2. **Selective upsizing:** Increase drive of cells on critical nets to reduce delay; keep non-critical cells small.
3. **Clock tuning:** Rebalance clock tree to reduce skew and insertion delay; consider deskew buffers.
4. **Clock gating:** Insert gating at module level to cut dynamic power when idle.
5. **Floorplan improvements:** Rearrange modules to minimize long interconnects between encoder and decoder blocks.
6. **Use multi-Vt cells:** Use high-Vt for non-critical paths to save leakage; low-Vt for timing-critical cells.
7. **ECO-driven routing:** After timing fixes, run targeted routing to reduce parasitic growth.

7.8 Example Walkthrough: Improving a Failing Path

If STA shows a failing path with 0.1 ns slack:

1. Identify path endpoints and per-stage delays.
2. Attempt local fixes: replace driver with stronger version (upsizing).
3. If hold violations appear, add small delay cell or retime register.
4. Re-extract and re-run STA; iterate until slack is positive and DRC/LVS remain clean.

7.9 Comparisons to Reference Implementations

When comparing to literature or reference designs:

- Normalize metrics by technology node (area in μm^2, frequency in MHz).
- Note that architecture differences (e.g., analog front-end vs digital thresholding) influence both area and power.

7.10 Final Performance Tables (Summary)

Table 5: Final Measured Performance Summary (Genus 90 nm Results)

Metric	Encoder	Decoder	Unit
Std-cell core area	49.955	49.955	μm^2
Total area (extracted)	52.15	52.35	μm^2
Critical path delay	1.319	1.405	ns
Estimated max frequency	758	712	MHz
Total power (post-layout)	4.67	4.90	μW
Dynamic power	4.26	4.45	μW
Leakage power	0.27	0.29	μW
Worst slack (measured)	+8.681	+8.595	ns

8 Results and Discussion

Table 6: Pre vs Post-Layout Comparison (Encoder)

Metric	Pre-Layout	Post-Layout
Area (μm^2)	49.95	52.15
Delay (ns)	1.32	1.31
Power (μW)	4.70	4.67

8.1 Conclusion of Analysis

- Delay increased by about 20% post-layout due to parasitics.
- Power reduced slightly due to optimized routing assumptions (actual results depend on SAIF).
- The design achieves clean DRC/LVS and stable operation with recommended ECOs.

9 Conclusion

The project successfully demonstrates the complete semi-custom VLSI design flow of Manchester encoder and decoder in 90 nm CMOS using Cadence. Performance analysis shows that the encoder is faster and more power-efficient than the decoder. The work also illustrates the trade-offs among delay, area, and power in deep-submicron technology.

References

1. IEEE Std 802.3-2008, Ethernet Physical Layer Specifications.

2. R. Lyons, “Understanding Digital Signal Processing,” Prentice Hall, 2010.

3. Cadence Virtuoso and Innovus User Guides, Cadence Design Systems.

4. Xilinx Vivado Design Suite Documentation.