

Research Migration Project

<https://esim.fossee.in/research-migration-project>



The Research Migration Project is an initiative of FOSSEE, IIT Bombay that promotes the use of eSim for reproducing published research circuits originally implemented using proprietary simulation tools. The objective is to migrate these validated designs to eSim to build an open source resource database.

Name of the participant: Meghna Pradeep

Affiliation / Institution: Department of Electronics and Communication Engineering, Muthoot Institute of Technology and Science (MITS), Puthencruz, Ernakulam, Kerala, India

Title of the circuit: Design and Simulation of an 8-bit DPCM Encoder Circuit

Software Used: eSim v2.5, KiCad Schematic Capture, Ngspice

Theory / Description

Differential Pulse Code Modulation (DPCM) is a predictive coding technique in which the difference between a present sample and a previously stored sample is encoded instead of directly encoding the sample value. This project reproduces the DPCM encoder core relevant to the Green (G) and Blue (B) image planes of an 8-bit RGB image described in the selected reference publication. The encoder is migrated to the open-source eSim environment using KiCad-based schematic capture and Ngspice simulation.

The circuit consists of an 8-bit previous-sample register implemented using eight clocked D flip-flops and an 8-bit ripple-borrow subtractor implemented using eight full-subtractor stages. Pixel-bit inputs are applied using PWL voltage sources, converted to digital signals using ADC bridges, processed by the digital subtractor, and converted back to analog voltages using DAC bridges for waveform observation.

A representative five-pixel stream $G = [100, 102, 101, 103, 103]$ is used for functional verification. The expected DPCM sequence is $[+100, +2, -1, +2, 0]$, corresponding to the 8-bit output bus values $[100, 2, 255, 2, 0]$, where 1111111_2 equals 255 when interpreted as unsigned but represents -1 in 8-bit two's-complement representation.

Introduction and Background

Image compression is important where image data must be stored or transmitted using limited bandwidth, such as in wireless capsule endoscopy. An RGB image consists of Red, Green, and Blue planes; the reference publication identifies strong correlation between neighboring pixels in the Green and Blue planes. Since neighboring pixel values tend to be similar, DPCM exploits this correlation by encoding the difference between consecutive samples instead of each sample independently.

For a sequence of samples $X[n]$, the previous sample is used as the prediction:

$$\hat{X}[n] = X[n-1] \quad D[n] = \hat{X}[n] - X[n-1]$$

where $X[n]$ is the current sample, $X[n-1]$ is the previously stored sample, and $D[n]$ is the DPCM difference. Highly correlated neighboring pixels produce small difference values that later compression stages (e.g. Huffman coding, outside this scope) can encode efficiently.

Theory and Operation

The DPCM encoder performs two functions: storing the previous sample and subtracting it from the current sample. The current pixel $X[n]$ is supplied to the circuit while $X[n-1]$ is stored in an 8-bit register of eight D flip-flops. At each active clock edge the current pixel is captured by the register, and the stored value becomes the previous sample for the following cycle. The subtractor receives $X[n]$, $X[n-1]$, and an initial borrow input ($\text{Bin} = 0$), generating the 8-bit DPCM output.

For the testbench, the previous value is initialized to zero, so $D[0] = 100 - 0 = 100$; subsequently $D[1] = 102 - 100 = 2$,

and $D[2]$
 $= 101 - 102 = -1$.

Two's-Complement Interpretation

Because the circuit produces an 8-bit binary result, negative differences must be interpreted appropriately: in 8-bit two's-complement, $-1 = 11111111_2$, which if read as unsigned equals 255. For arbitrary unsigned 8-bit pixels, subtraction can mathematically produce values from -255 to $+255$, so an 8-bit signed representation cannot cover every possible difference. The

-127 to $+128$ range reported in the reference paper is the observed output range for its tested image data, and should not be confused with the conventional signed range of an 8-bit two's-complement number.

Binary Output	Unsigned Interpretation	Signed 8-bit Interpretation
01100100	100	+100
00000010	2	+2
11111111	255	-1
00000000	0	0

Reason to reproduce with eSim

The reference design's hardware is reported using commercial (Cadence-class) EDA tool-sets. Migrating the DPCM encoder core to eSim/Ngspice gives an open-source, reproducible, component-level implementation using standard digital blocks (d_dff, full_sub, half_sub) together with mixed-signal ADC/DAC bridges already supported in eSim, and lets the design's correctness be verified directly against the paper's reported behaviour — at no licensing cost and with full transparency of every node and subcircuit.

Reference Work	Present eSim Project
8-bit RGB image	8-bit pixel input
Green and Blue planes	G/B DPCM concept retained
Previous pixel used for prediction	$8 \times$ D flip-flop previous-sample register
Difference calculation	8-bit ripple-borrow subtractor
Clock-managed previous data	Common clock for DFF register
DPCM output	8-bit subtractor output bus
Hardware implementation	eSim circuit implementation
Commercial/hardware EDA environment	Open-source eSim / Ngspice environment

Expected Outcome/outputs

- Correct storage of the previous 8-bit pixel $X[n-1]$ in the D flip-flop register on every clock edge.
- Correct 8-bit ripple-borrow subtraction $D[n] = X[n] - X[n-1]$ with borrow propagating from bit 0 through bit 7.
- Signed two's-complement DPCM output verified against a hand-worked 5-pixel test sequence.
- Zero output for identical consecutive pixels; small-magnitude output for slowly varying pixels; correct borrow-out assertion for negative differences.
- Bit-level verification via the eSim Python Plotter and Ngspice console, plus bus-level verification via decimal reconstruction of the 8-bit output.

Circuit Diagram(s)

The complete component-level schematic was captured in eSim's KiCad-based schematic editor. It shows all component designators, subcircuit instances, ADC/DAC bridge models, and node-level wiring exactly as simulated.

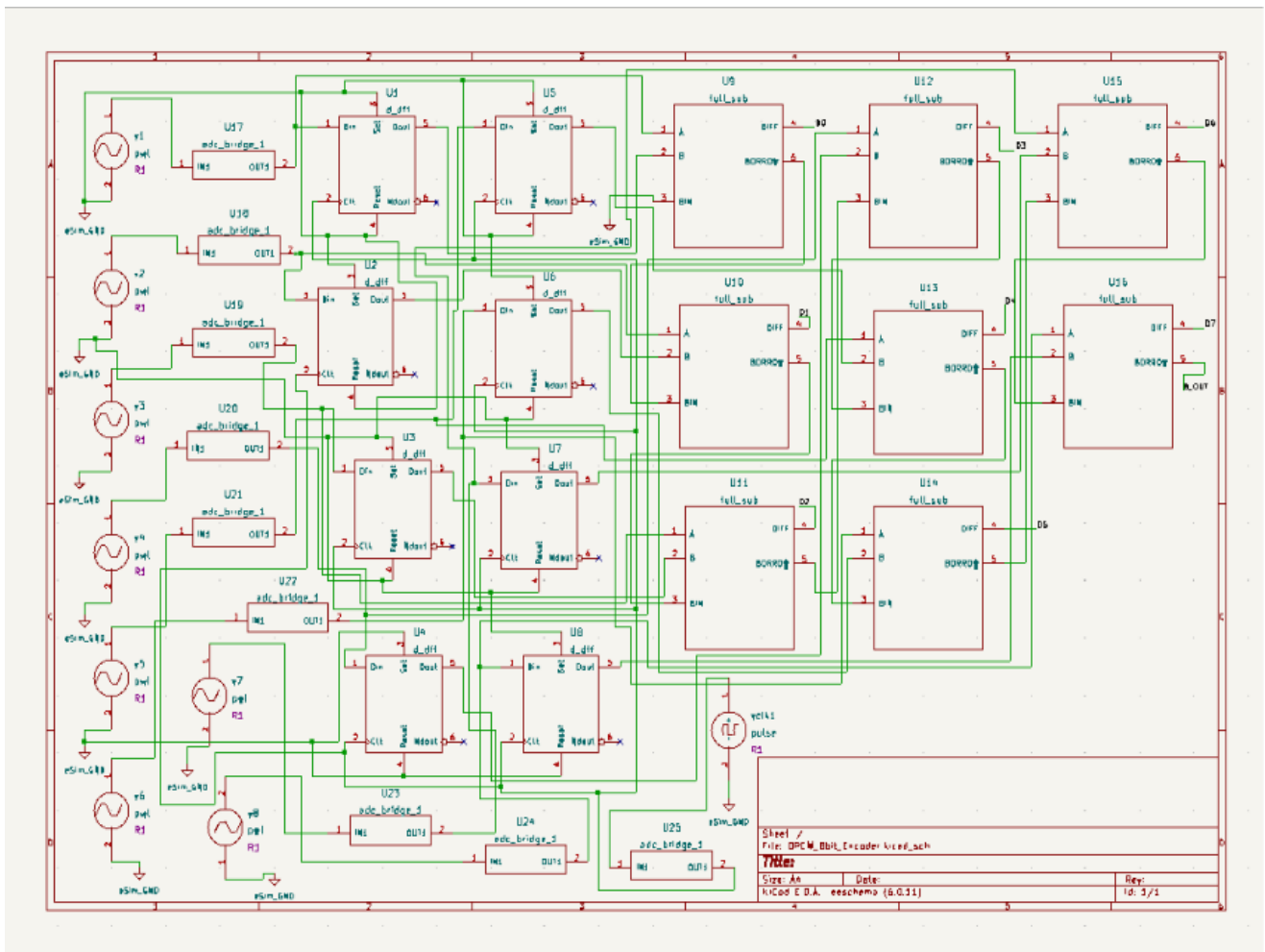


Figure 1. Complete eSim/KiCad schematic of the 8-bit DPCM encoder — PWL sources V1–V8 ($\hat{X}[n]$) with ADC bridges U17–U25, D flip-flop register U1–U8, cascaded full-subtractor chain U9–U16, and DAC-bridged difference/borrow outputs D0–D7 and Bout.

ADC Bridges: PWL analog voltages are converted into XSPICE digital logic events using .model adc_bridge adc_bridge(in_low=1.0 in_high=2.0). DAC Bridges: digital outputs /d0–/d7 and /b_out are converted back to continuous analog voltages v_d0–v_d7 and v_bout using .model dac_bridge dac_bridge(out_low=0.0 out_high=5.0). Digital ground (d_gnd) is tied to analog ground through a 0V reference to guarantee simulator matrix convergence.

Block Diagram(s)

Original overall block diagram:

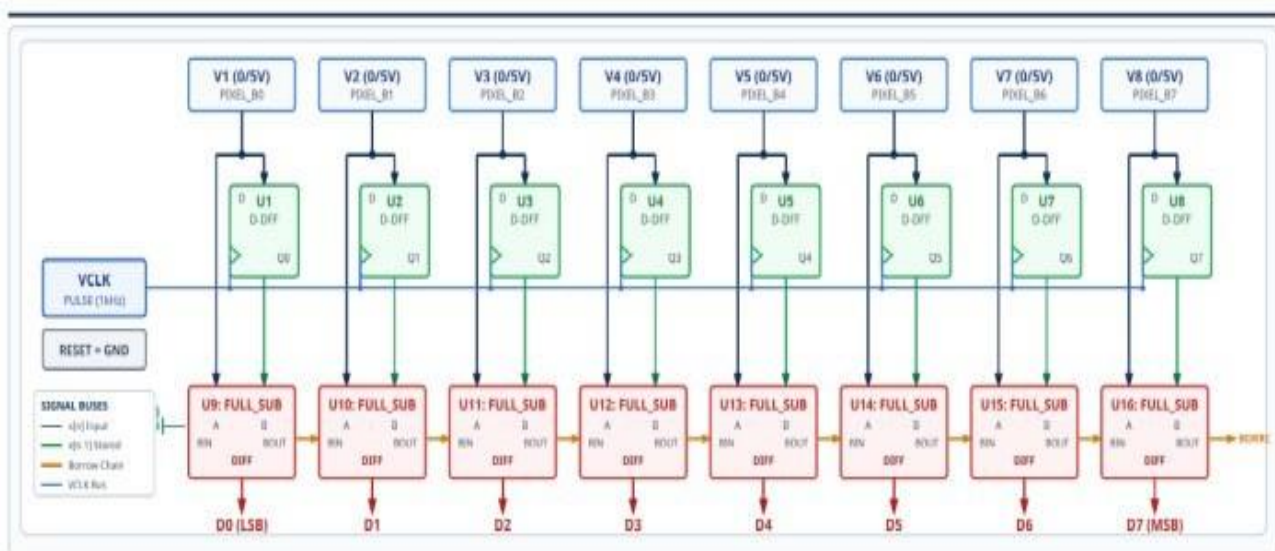


Figure 2. Top-level hardware block diagram of the 8-bit DPCM encoder: input sources V1–V8, clock VCLK, D flip-flops U1–U8, full subtractors U9–U16, difference outputs D0–D7, and borrow-out.

Eight independent voltage sources (V1–V8) provide the parallel 8-bit pixel word $X[n]$ (X0/LSB to X7/MSB). A synchronous 1kHz clock (VCLK) is distributed to all eight positive-edge-triggered D flip-flops (U1–U8). $X[n]$ is routed simultaneously to the minuend inputs (A) of full subtractors U9–U16 and to the D inputs of the register. The register outputs (Q0–Q7) hold $X[n-1]$ and feed the subtrahend inputs (B). The ripple-borrow chain begins at U9 with Bin tied to 0V and propagates serially (Bout→Bin) through to U16, producing the 8-bit difference word D0–D7 and final borrow Bout.

One-bit Full Subtractor (full_sub) and Half Subtractor (half_sub)

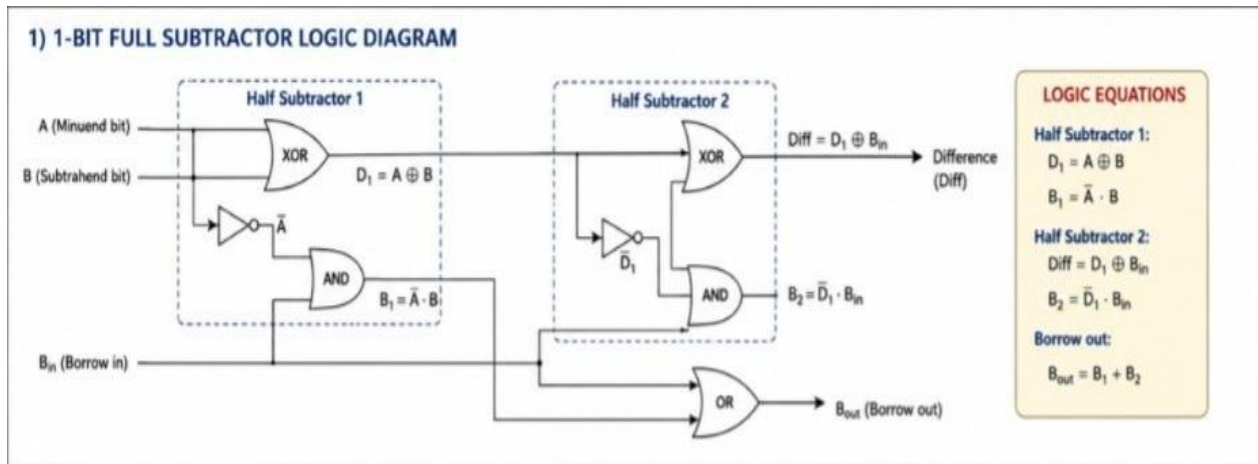
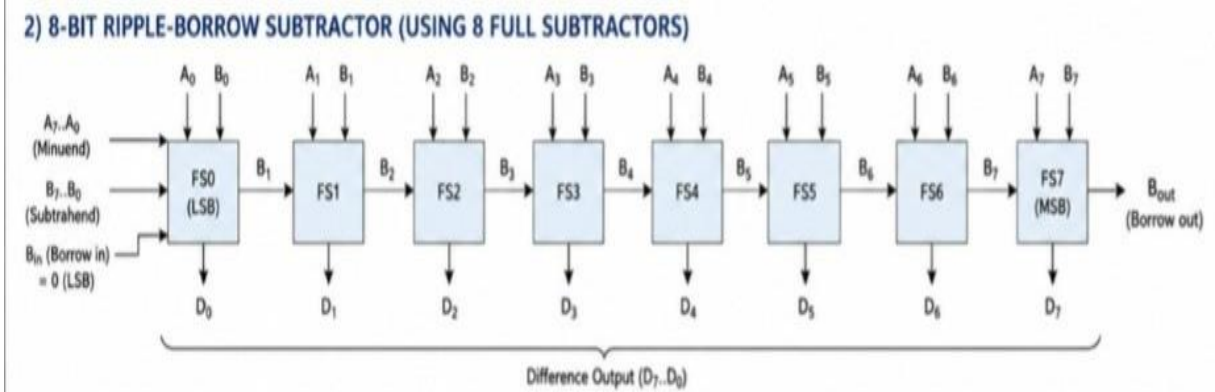


Figure 3. Gate-level implementation of the 1-bit full subtractor using two half-subtractor stages and an OR gate, corresponding to full_sub.sub and half_sub.sub.

Half Subtractor 1 receives A and B, generating intermediate difference $D_1 = A \oplus B$ and intermediate borrow $B_1 = \bar{A} \cdot B$. Half Subtractor 2 receives D_1 and B_{in} , computing the final difference $Diff = D_1 \oplus B_{in}$ and second intermediate borrow $B_2 = \bar{D_1} \cdot B_{in}$. An OR gate combines the intermediate borrows: $B_{out} = B_1 + B_2$.

8-bit Ripple-Borrow Subtractor

Figure 4. 8-bit ripple-borrow subtractor showing the cascaded borrow chain across stages FS0 to FS7 (U9 to U16).



Eight 1-bit full-subtractor modules (FS0–FS7, instantiated as U9–U16) are connected in cascade to form the 8-bit arithmetic difference block. FS0 processes the LSBs (A_0, B_0) with initial borrow $B_{in} = 0$. Each stage's borrow-out feeds the borrow-in of the next higher-order stage; FS7 processes the MSBs (A_7, B_7) and outputs the terminal borrow Bout. The individual Diff pins together form the 8-bit parallel bus $D_7 \dots D_0$.

8-bit Previous-Sample Register

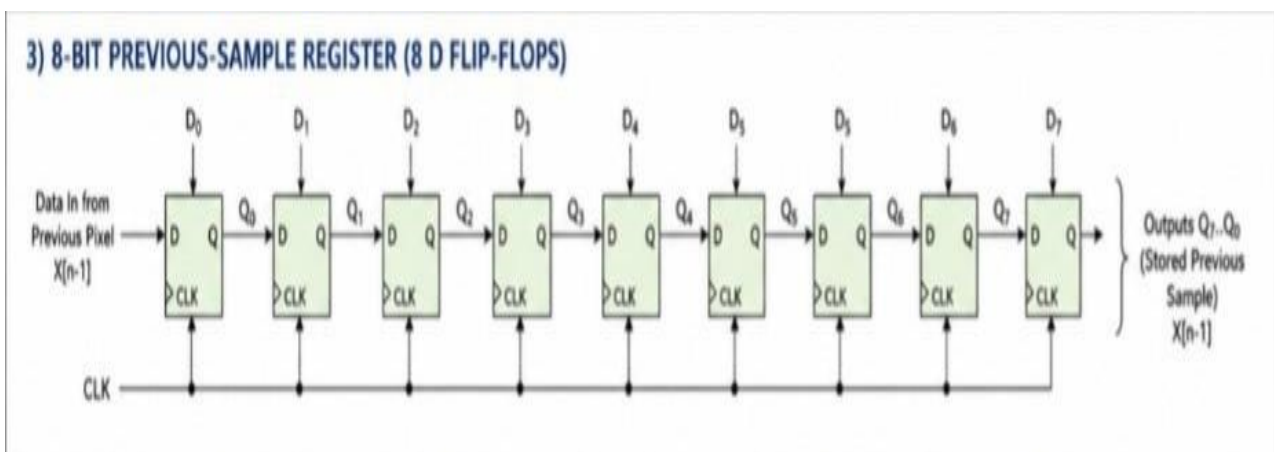


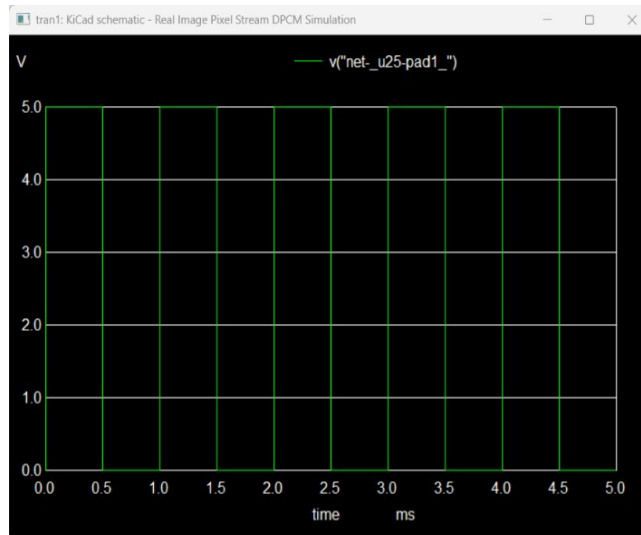
Figure 5. Synchronous 8-bit register bank implemented using eight D flip-flops (U1–U8) driven by the common clock VCLK.

This bank implements the unit-delay prediction $\hat{X}[n] = X[n-1]$. Eight positive-edge-triggered D flip-flops receive input bits D0–D7 from the incoming pixel bus, sharing the 1kHz system clock on their CLK pins. On every rising edge the active input byte is latched onto Q0–Q7, holding the prior pixel data stable across the 1.0ms period so ripple-borrow subtraction can settle.

Results (Input, Output waveforms and/or Multimeter readings)

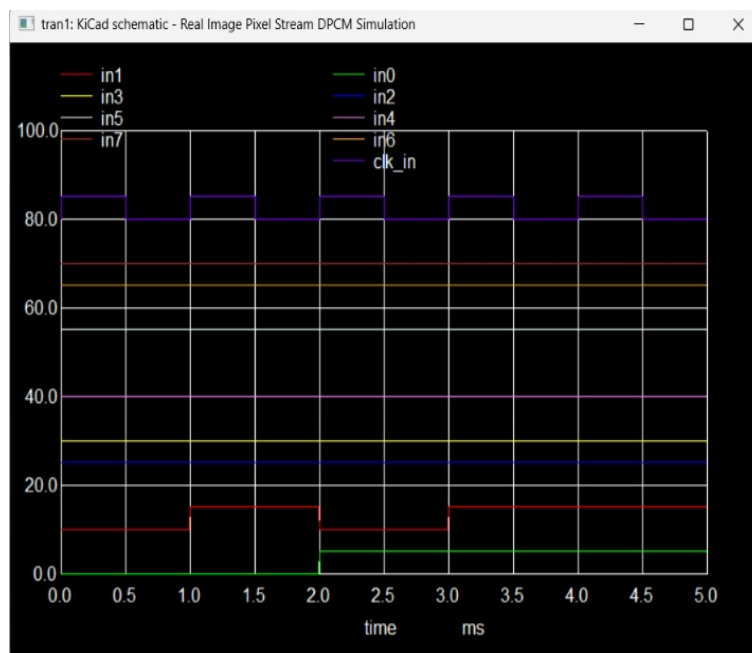
The 8-bit DPCM encoder was simulated in eSim v2.5 using mixed-signal Ngspice transient analysis over a 5.0ms duration (1.0μs max time step). Verification was performed at the bit level using the eSim Python Plotter and vertically offset Ngspice console waveforms, and at the bus level via decimal reconstruction.

Figure 6 — System Clock Waveform



$v('net_u25-pad1_')$, source VCLK1: a clean 0–5V periodic square pulse, 1kHz ($T = 1.0ms$), 500μs pulse width, 50% duty cycle. The positive edges latch $\hat{X}[n]$ into the register bank on each of the five 1ms cycles.

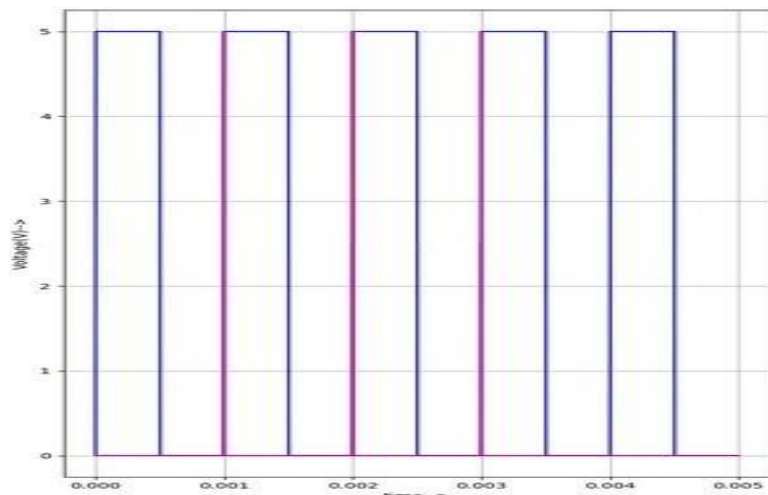
Figure 7— Applied 8-bit Input Pixel Waveforms



$in0$ – $in7$ (10V vertical offsets) with clk_in . Bits 2, 5, 6 stay high (weighted sum $4+32+64=100$, the common base intensity); bits 0 and 1 step to encode the sequence $100 \rightarrow 102 \rightarrow 101 \rightarrow 103 \rightarrow 103$ (binary $01100100 \rightarrow 01100110 \rightarrow 01100101 \rightarrow 01100111 \rightarrow 01100111$).

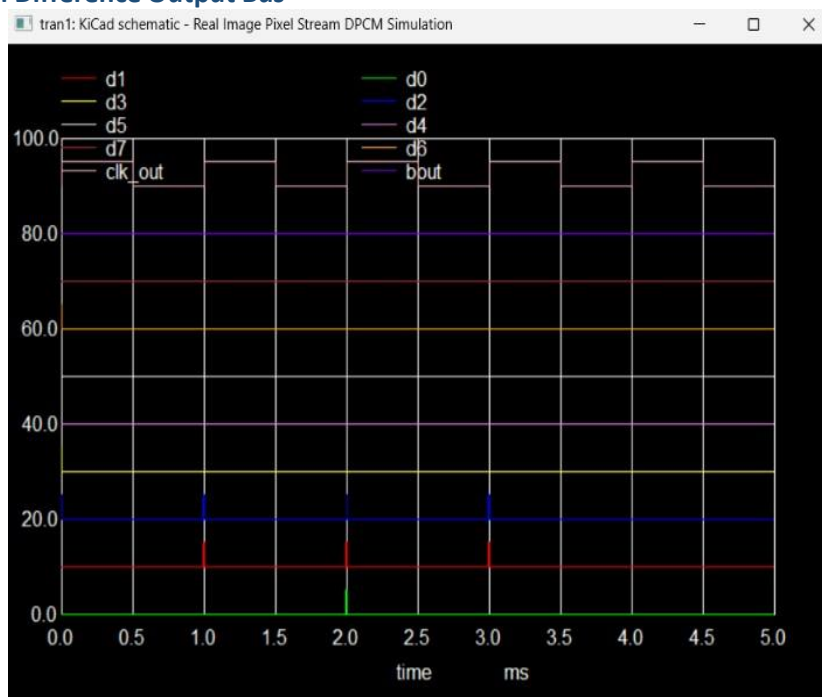
Figure 8 — eSim Python Plotter Verification Waveform

Clock (blue), difference bit v_d1 (magenta), borrow-out v_bout (yellow). Cycles 2 & 4: v_d1 high, v_bout low $\rightarrow +2$. Cycle 3: both v_d1 and



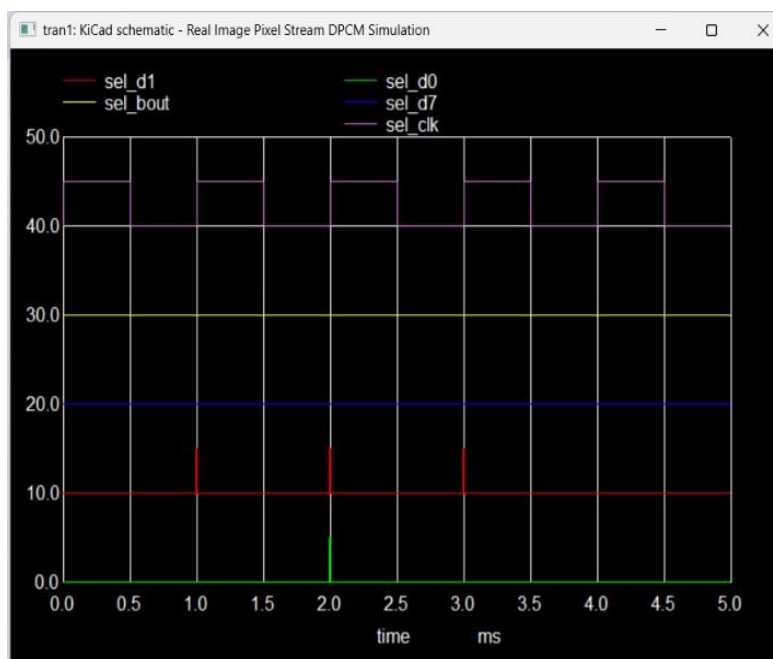
v_bout assert $\rightarrow$ borrow generated for -1 . Cycle 5: v_d1 stays at $0V \rightarrow$ zero difference.

Figure 9 — Complete 8-bit DPCM Difference Output Bus



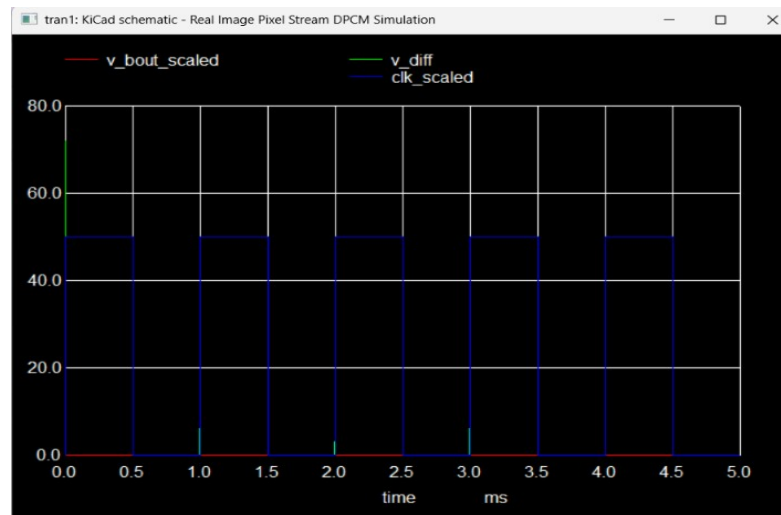
d0–d7, bout, clk_out (10V offsets). Cycle 1: d2,d5,d6 high (100). Cycle 2: d1 high (+2). Cycle 3: all d0–d7 and bout high (11111111 = 255 unsigned = -1 signed). Cycle 4: d1 high again (+2). Cycle 5: all outputs flat (0).

Figure 10 — Selected Output Difference Bits and Borrow



sel_d0, sel_d1, sel_d7, sel_bout, sel_clk isolate the critical bits: sel_d0 confirms the odd-valued difference in Cycle 3; sel_d1 confirms +2 in Cycles 2 and 4; sel_d7/sel_bout confirm the negative-delta borrow in Cycle 3 only.

Figure 11 — Reconstructed Decimal DPCM Bus Waveform



Weighted reconstruction $v_diff = \sum(v_di/5) \cdot 2^i$ plotted with v_bout_scaled and clk_scaled . Green trace steps 100→2→255→2→0, numerically confirming the encoder's per-cycle output against the theoretical pixel calculations.

Cycle-by-Cycle Verification Table

Cycle	Time (ms)	X[n]	X[n-1]	D[n]	Binary Output	Unsigned Bus	Bout	State
1	0.0–1.0	100 (01100100)	0 (00000000)	+100	01100100	100	0	Initial storage
2	1.0–2.0	102 (01100110)	100 (01100100)	+2	00000010	2	0	Positive delta
3	2.0–3.0	101 (01100101)	102 (01100110)	-1	11111111	255	1	Two's-complement (-1)
4	3.0–4.0	103 (01100111)	101 (01100101)	+2	00000010	2	0	Positive delta
5	4.0–5.0	103 (01100111)	103 (01100111)	0	00000000	0	0	Identical input, zero delta

Simulation and Reproducibility Note:

The migrated circuit was successfully simulated in eSim v2.5 using Ngspice with explicit XSPICE ADC/DAC bridge model definitions. The supplied simulation configuration **should be run directly without regenerating the KiCad-to-Ngspice output**, as reconversion may overwrite the required bridge-model definitions and prevent reproduction of the reported results. Verification was performed primarily through Ngspice console waveforms, including vertically offset bit-level plots and decimal bus reconstruction, with additional critical-bit verification using the eSim Python Plotter.

Research Paper/Journal/etc.

Title: An Ingenious Design of a High Performance-Low Complexity Image Compressor for Wireless Capsule Endoscopy

Author: Ioannis Intzes, Hongying Meng, John Cosmas

Page No.: Sensors, vol. 20, no. 6, p. 1617, 2020

Link: <https://doi.org/10.3390/s20061617>

The reference work proposes an image-compression architecture for wireless capsule endoscopy, applying DPCM to the Green and Blue planes of an 8-bit RGB image because neighboring pixels in these planes are strongly correlated. It reports an observed DPCM output range of -127 to +128 for its tested image data. The complete reference architecture also includes Huffman coding, serialization, and hardware implementation; these remain outside the scope of the present migration, which concentrates on the DPCM encoder core.

Source/Reference(s)

- I. Intzes, H. Meng, and J. Cosmas, "An Ingenious Design of a High Performance-Low Complexity Image Compressor for Wireless Capsule Endoscopy," Sensors, vol. 20, no. 6, p. 1617, 2020. DOI:

10.3390/s20061617.

- FOSSEE, IIT Bombay, “eSim: An Open Source EDA Tool for Circuit Design, Simulation, Analysis and PCB Design,” User Manual & Reference Documentation, v2.5.

Note: Fields marked with an asterisk () are mandatory and must be filled for successful submission.*