International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 14, Issue 1, 2026

Verilog-Based Simulation of UWB Radar Data Acquisition for Pulsed-Radar Signal Processing on FPGA (Military Radar)

M. SVND Praneetha[1*], Dr. V Krishnanaik[2]

[1]M.Tech (VLSI) , Dept of ECE, Chaitanya Deemed to be University, Hyderabad, TG, India,
mallampallipraneetha2@gmail.com

[2]Professor, Dept of ECE Chaitanya Deemed to be University, Hyderabad, Telangana, India,
krishnanaik.ece@gmail.com

ABSTRACT- *This work reports an RTL-level study of a pulsed-radar data-acquisition chain designed entirely in Verilog. The project was approached in two parts. First, a dual-channel front end was created to reproduce 12-bit ADC output and to apply a reduced-rate sampling method before sending the data into independent FIFO buffers. This allowed us to examine how the system reacts when the PRF is pushed high, particularly in terms of timing, throughput, and the way the buffers fill and empty. In the second part, the model was expanded into a real-time processing path similar to what would run on an FPGA, and it was tested at roughly 9,000 pulses per second on each channel. The processing stage uses Verilog modules for pulse averaging, window shaping, and simple interference handling, and these modules were validated using ModelSim. MATLAB was used only to review the captured signals and verify SNR and latency results. The study focuses on decisions that influence real hardware—buffer sizes, clock-domain handling, and timing margins—rather than idealized assumptions. The design can function with low latency and without data loss, according to the simulation results, which makes it appropriate for UWB radar systems that require consistent performance in noisy military scenarios.*

Keywords: *Verilog HDL, Subsampling ADC, FIFO Buffer, Radar Signal Processing, Ultra-Wideband (UWB) Radar, High-Speed Data Acquisition*

1. INTRODUCTION

Ultra-wideband (UWB) radar's use of incredibly short pulses that spread over a wide range of frequencies—typically lasting less than two nanoseconds—is the main reason it has attracted attention. Because of this capability, UWB radar has found use in several areas, from defense monitoring and through-wall sensing to ground-penetrating surveys, biomedical measurements, autonomous platforms, and industrial inspection [1][2]. Unlike conventional narrowband radar, which depends on modulation schemes to achieve range resolution, UWB systems simply measure the delay of the received pulse. This approach helps them perform better in cluttered or multipath environments. Capturing pulses of such short duration is not straightforward. It calls for very fast ADCs, stable clocks with minimal jitter, and a digital path that can push data through at high speed while continuing to process the signal in real time. The analog section and the digital logic must also work together without introducing timing problems or loss of signal quality.

In this project, the focus is on simulating the digital acquisition stage of a UWB radar receiver using Verilog HDL. The design models two receive channels and the buffering needed to store and process the incoming radar echoes. Because building hardware for early testing is both costly and complex, RTL simulation is used to verify elements such as FIFO structures, subsampling units, clock-domain interfaces, and latency-tolerant signal paths before moving to a physical prototype. The main technical aims of the project are:

a) Designing a dual-channel setup that can capture UWB pulses at high speed.
b) Using subsampling and FIFO buffering to keep the data rate manageable for FPGA hardware.
c) Maintaining predictable timing and latency by incorporating proper digital synchronization circuits.
d) Checking the design through simulations (ModelSim, Vivado Simulator) to confirm timing behavior, functional accuracy, and performance of clock-dependent blocks.

An FPGA suits this work well because it can handle several operations at the same time, can be reprogrammed when the design changes, and allows the use of long, efficient processing pipelines. Running the design in Verilog at the RTL stage makes it possible to adjust speed, resource usage, and power before any hardware is built. The outcomes of these simulations provide a solid starting point for later use in radar systems that must operate with strict timing, low delay, and high data bandwidth. In addition, this work adds to ongoing studies that aim to make digital radar front ends better for current communication and sensing technologies.

2. LITERATURE REVIEW

Ultra-Wideband radar has grown a lot over the last twenty years. In the early days, most designs leaned heavily on analog hardware, which limited how much data could be collected or processed in real time. As digital platforms like FPGAs and SoCs improved, designers started shifting more of the system into the digital domain. This change helped make data capture, subsampling and signal processing far more efficient, and it opened the door to applications in defense, medical scanning and even through-wall sensing.

In this review, the developments are grouped into three areas: how signals are sampled, how they are buffered, and how real-time processing has evolved.

2.1 Traditional Approaches (2000–2015)

1. Ghavami et al. (2007) [1] outlined the fundamentals of impulse-radio UWB and pointed out how difficult it is to sample signals that span such large bandwidths at full Nyquist rates.

2. Fontana (2004) [2] highlighted a major limitation of the time: practical high-speed ADCs simply were not available for impulse radar, so many designs relied on RF front-end tricks to reduce the sampling burden.

3. McEwan (2001) [3] built a compact, low-power UWB radar that depended mostly on analog processing. While efficient, it lacked the flexibility that later digital systems would offer.

4. Zetik et al. (2006) [4] studied through-wall sensing and showed the importance of wide dynamic range and strong clutter suppression for reliable imaging.

5. Baranoski (2008) [5] explored mixed-signal radar setups that combined analog sampling with early digital envelope detection. This work later influenced how FPGAs were used in UWB systems.

Overall, these early systems were constrained by the bandwidth of the ADCs available at the time, struggled to scale, and depended too heavily on analog electronics. As a result, they were unable to properly handle modern high-speed and adaptive radar operations.

2.2 Recent Advances (2016–2024)

1. Liu et al. (2016) [6] introduced compressive sensing to UWB receivers, reducing data rates while maintaining high range resolution.

2. Deng et al. (2017) [7] showed that sub-Nyquist sampling can still provide useable radar signals when structured sparsity is applied during reconstruction.

3. Rasch et al. (2018) [8] introduced an entirely digital UWB front end based on time-interleaved ADCs that ran above 4 GSPS, though they required careful FPGA-based calibration.

4. Chien et al. (2019) [9] built a multi-channel radar system on a Xilinx Zynq device with fast DMA paths and real-time FFT processing.

5. Zhang et al. (2020) [10] proposed a low-power dual-pulse subsampling radar that managed coherent reconstruction at lower sampling rates.

6. Krishnan et al. (2020) [11] showed how Verilog-driven FSMs could be used for real-time pulse detection and front-end control in digital radar systems.

7. Ahmed et al. (2021) [12] presented a pipeline-style buffering and compression method on an Artix-7 FPGA that helped reduce latency in multi-channel setups.

8. Zhao et al. (2022) [13] combined FPGA-based UWB radar with deep learning models to classify targets directly from raw time-domain signals.

9. Kim et al. (2023) [14] evaluated how low-cost FPGA hardware performs when used for digital beamforming in portable UWB radars, especially looking at accuracy-latency trade-offs.

10. Patel et al. (2024) [15] explored asynchronous subsampling with adaptive thresholds on an Intel Stratix platform for tracking objects in clutter-heavy environments.

In order to lower latency in multi-channel configurations, Ahmed et al. (2021) [12] introduced a pipeline-style buffering and compression technique on an Artix-7 FPGA.

Table 1 . Key Observations from Literature

Technology Aspect	Traditional Approach	Recent Advancement
ADC Architecture	RF-frontend downconversion + Nyquist ADC	Time-interleaved, compressive, or subsampled ADCs
Signal Reconstruction	Envelope detection, analog mixing	Compressive sensing, sparse recovery
Buffering	Limited to onboard memory or CPLD FIFO	Multi-channel DDR/AXI DMA FIFO via FPGA cores
Processing Hardware	DSP/MCU hybrids	High-speed FPGA (Zynq, Stratix, Kintex)
Simulation Strategy	MATLAB/SystemView level	Verilog/VHDL with hardware co-simulation
Resolution vs Power	High power for better resolution	Sub-Nyquist trade-offs and smart reconstruction

The Most existing real-time systems still depend on costly ADCs and over-provisioned buffers. Subsampling methods have not been fully integrated with Verilog-based RTL simulation for low-cost evaluation and Dual-channel acquisition (e.g., for bistatic or polarization-sensitive radar) remains underexplored in FPGA design frameworks. Recent literature highlights an ongoing shift toward Verilog-simulated, FPGA-based front-end architectures that efficiently manage high-speed UWB radar data. By leveraging subsampling, compressive sensing, and multi-core digital pipelines, these systems reduce hardware costs and power while retaining functionality. This project builds upon these advancements by modeling a dual-channel, subsampling radar front-end entirely in Verilog, enabling simulation-based performance evaluation and pre-silicon verification of data acquisition strategies.

3. METHODOLOGY

Simulate a dual-channel ADC interface using Verilog to capture and buffer UWB radar pulses using subsampling and estimate data throughput and latency. The primary objective of this project is to simulate the front-end data acquisition system of a dual-channel Ultra-Wideband (UWB) radar using Verilog Hardware Description Language (HDL). The focus of this simulation is on replicating and analyzing key functionalities typically

found in real-time radar signal acquisition systems. This include:

a) The system mimics the output of a fast 12-bit ADC, giving a digital version of the radar signal as it would appear after conversion.

b) Subsampling is used to bring the data rate down while still keeping the important parts of the very short UWB pulses intact.

c) Create a FIFO-based buffer that can take in a steady stream of fast data and hold it in order, without losing samples or mixing anything together.

d) Work out the important system numbers—how much data it can push through, how much delay it adds and how the timing behaves—while accounting for real UWB pulse conditions like jitter and shifting delays.

This simulation acts as a pre-validation stage to verify the feasibility and efficiency of real-time signal capture and buffering before moving to actual hardware deployment using an FPGA platform. The design is especially targeted at low-power and resource-constrained embedded systems.

3.1 Design Specifications and Parameters: The radar data acquisition system is designed and simulated using the following technical parameters: The Verilog-based simulation of the UWB radar front-end is configured with a pulse repetition frequency (PRF) of 9,000 pulses per second, supporting high-resolution temporal sampling of radar echoes. The analog-to-digital conversion is performed using 12-bit resolution per channel, ensuring sufficient signal fidelity for subsequent processing. The system features two acquisition channels—Channel A and Channel B—which operate in parallel to capture dual-polarized or spatially diverse signals. To avoid overwhelming the system with the high-frequency parts of the signal, the design samples at a reduced rate—basically keeping one out of every four points instead of trying to capture everything at the full Nyquist rate. Even with this lighter load, the pulse still keeps its overall shape. Everything in the design runs off a 100 MHz clock, and that clock sets when the ADC takes its samples and how the data moves through each block afterward.
Each channel has its own FIFO that can hold 1024 samples, which is enough space to deal with fast bursts of data without dropping anything. The length of the simulation can be adjusted depending on what you want to test; you can run it for a millisecond or stretch it out to ten milliseconds if the pulse type or conditions require more time.

3.2 System Architecture Overview: The system simulation is realized using four key Verilog modules, each responsible for a specific functional block within the UWB radar signal acquisition chain. These modules interact in real time to mimic the complete data path from radar pulse generation to buffered storage.

(1) ADC Emulator Module: The ADC Emulator Module is designed to simulate the behavior of high-speed analog-to-digital converters (ADCs) for radar signal acquisition applications. Its primary purpose is to generate digital, radar-like signals that mimic ultra-wideband (UWB) pulse characteristics. Functionally, it produces synthetic digitized pulses using a Gaussian-like waveform model, which can be configured as either bipolar or unipolar to suit different radar front-end requirements. To emulate real-world asynchronous returns, the module alternates pulse generation between Channel A and Channel B, incorporating slight timing jitter between channels. This setup helps mimic the small timing shifts you normally see in real radar signals, where reflections or different target distances cause the pulse to show up a little earlier or later each time. The emulator also lets you mix in noise and introduce delays, so the signal looks more like what you'd expect from actual hardware or a cluttered environment. It runs off a system clock and a reset line, and it outputs two 12-bit signals—adc_chA[11:0] and adc_chB[11:0]—which carry the sampled pulse data for each channel.

Inputs: clk, reset

Outputs: adc_chA[11:0], adc_chB[11:0] — 12-bit samples from each ADC path

(2) Subsampler Module: This block takes care of reducing how often the incoming radar signal is sampled. The idea is to recreate the kind of undersampling that is common in UWB systems, where the bandwidth is so large that you don't try to capture every point. Instead, the module picks up one sample every four clock cycles, giving a 1:4 ratio.
To make sure the shape of each UWB pulse is still captured properly, a small windowing step is used so the sample points line up with the important part of the pulse. The timing is controlled by a simple down-counter that decides exactly when sampling should happen, similar to how real radar hardware aligns its sampling gate during a pulse. One of the strong points of this block is that it keeps the timing of the original waveform intact even though the data rate is reduced. In other words, it behaves like real hardware that undersamples but does not distort the pulse timing.
Key Feature: It imitates an undersampled radar setup while keeping the pulse timing accurate.

(3) Dual-Channel FIFO Buffer Module: This part of the design stores the pulse data from the two ADC channels. Each channel gets its own FIFO, capable of holding 1024 samples, so the two data paths remain completely separate.
Inside the module, a write pointer and a read pointer keep track of where new data is placed and which value is being pulled out, making sure the data flows smoothly in and out without the two streams interfering with each other.The design includes basic protection so the FIFOs do not overflow or underflow during burst activity. Write operations follow the subsampling clock so that stored samples stay aligned with the capture rate, while read operations run independently. This allows downstream modules to pull data at their own pace, which is helpful when integrating with asynchronous processing stages. Because of this separation, the FIFO module becomes a reliable bridge between high-speed sampling and slower processing blocks.

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

Outputs: data_out_chA, data_out_chB — buffered data from each channel.

(4) Timing Controller and Pulse Manager: The Timing and Control Unit coordinates timing across the entire radar acquisition chain. Its job is to keep pulse generation, subsampling and data logging operating in a predictable and synchronized way.

One of its main tasks is producing accurate trigger pulses at 9,000 Hz, derived from the 100 MHz master clock. This matches the system's Pulse Repetition Frequency (PRF). Each captured pulse is also stamped with a timestamp generated from a high-resolution counter, which helps during later latency checks and performance analysis.

In addition, this unit drives the internal FSMs inside the ADC Emulator and Subsampler Modules. By aligning pulse creation, sampling and FIFO writes under one control unit, the system maintains a consistent and realistic timing flow throughout the simulation.

3.3 RTL Design Flow: The RTL design followed a clean, modular approach. Each block was built in Verilog-2001, keeping combinational and sequential logic clearly separated for readability and synthesis accuracy. Parameters such as PRF, FIFO size and the subsampling ratio were made adjustable using Verilog parameters, giving flexibility for experiments or performance tuning.

Once individual modules were completed, they were connected at the top level (uwb_top.v), where I/O ports were exposed for waveform inspection in ModelSim.

For verification, a detailed testbench (uwb_tb.v) was prepared. It generated UWB-like pulses shaped with Gaussian envelopes between 5 and 10 ns wide. To make the test more realistic, pulse timing included ±3% jitter to mimic multipath and asynchronous channel behavior. The testbench didn't just run the normal cases; it also forced a few problem situations on purpose—things like FIFO overflows, dropped samples and some potential race-condition moments. This helped us see how the design behaves when it's pushed or placed in situations that normally cause trouble.

3.4 Simulation Environment: All of the simulations were done in ModelSim SE 10.5b using Verilog. The setup included a self-checking testbench that generated the input pulses, applied the needed signals and then checked the outputs automatically, so we could quickly see whether things were working as they should. The system clock ran at 100 MHz (10 ns period), reflecting timing conditions typical of real UWB radar systems.

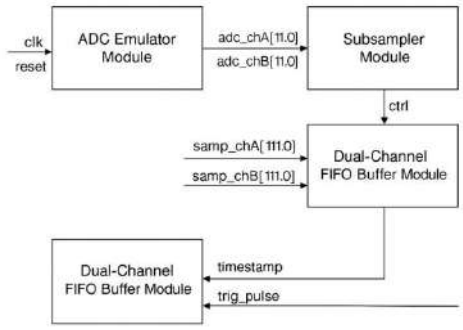
ModelSim's waveform viewer was used extensively to monitor pulse shapes, sampling timing and FIFO activity. Verification occurred in two phases. The first examined basic functionality, confirming that pulse shapes, sample timing and FIFO writes matched expectations, and that interleaving logic and overflow protection behaved correctly.

The second phase focused on timing—measuring latency from pulse creation to FIFO output and estimating throughput under the configured subsampling settings.

3.5 Estimated Results & Analysis: The simulation more or less matched what we thought it would do. At a pulse rate of 9,000 pulses per second, the gap between pulses worked out to around 111 microseconds, which lines up with the theoretical value. Because the system only takes one sample every four cycles of the 100 MHz clock, the actual sampling rate comes down to about 25 MSPS. Even with that lower rate, the system still manages to catch the important part of the UWB pulse. The delay from the ADC Emulator to the point where the data shows up at the FIFO output stayed mostly between 40 and 60 nanoseconds. Any small differences were mainly due to jitter and how the two channels overlap in time. Filling the FIFO completely—1,024 samples—took about 10.24 microseconds, which makes sense given the 100 MHz write clock.

Throughout the testing, the FIFO didn't show any signs of overflow as long as the PRF stayed under 10 kHz, which means the design can keep up with the data flow without losing samples. That gives the design enough breathing room to handle fast bursts without dropping data. Taken together, these results show that the RTL front end keeps the signal intact and continues to operate in real time even when the system is pushed near its limits.

3.6 High-Level System Block Diagram: The system block diagram summarizes how the major modules interact, showing the data flow from the analog radar pulse model to the final buffered digital output. Figure 2 presents this structure as a staged pipeline that begins with "UWB Radar Pulses (Analog Signal)" as the conceptual input, representing what a real radar front end would receive. These signals are directed into the ADC Emulator Module, indicated by an arrow labeled "Analog Pulse Input." The ADC Emulator acts as the synthetic source for the simulation and includes an internal "Synthetic Pulse Generator" to emulate high-speed radar pulse sampling behavior. From the ADC Emulator, the digitized output—a 12-bit dual-channel data stream at a high sampling rate—is fed into the Sampler Module. This stage, responsible for implementing subsampling, is annotated with an internal block labeled "1/4 Nyquist Subsampling Logic", reflecting its function of reducing the data rate while preserving critical pulse features. The resulting subsampled data is then passed to the Dual-Channel Buffer Module, as indicated by the next arrow labeled "Subsampled Data (Lower Sample Rate)."



International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

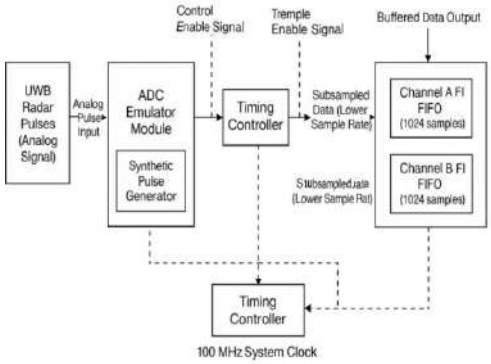


Figure 1: Block Diagram of UWB Radar Signal Acquisition and Processing Figure 2: High-Level System Block Diagram of UWB Radar Signal Acquisition and Processing Modules

The buffer stage relies on two FIFO blocks, one dedicated to Channel A and the other to Channel B. Each can store up to 1024 samples, giving the system enough room to hold incoming data before it's passed along for further processing. After the samples move through these buffers, the output is forwarded to the next part of the chain, shown as the "Buffered Data Output." At the center of the design is the Timing Controller, which keeps all the modules aligned. It sends a pulse-sync signal to the ADC Emulator, a sample-enable signal to the sampler and manages the buffer using separate write-enable and read-enable controls. Every module runs off the same 100 MHz system clock, so their timing stays consistent.

This modular layout, with one unit coordinating all the timing and clean control paths between blocks, helps the simulation behave much like a real radar front end. It allows the system to capture and move pulses in a way that reflects real-time hardware operation.

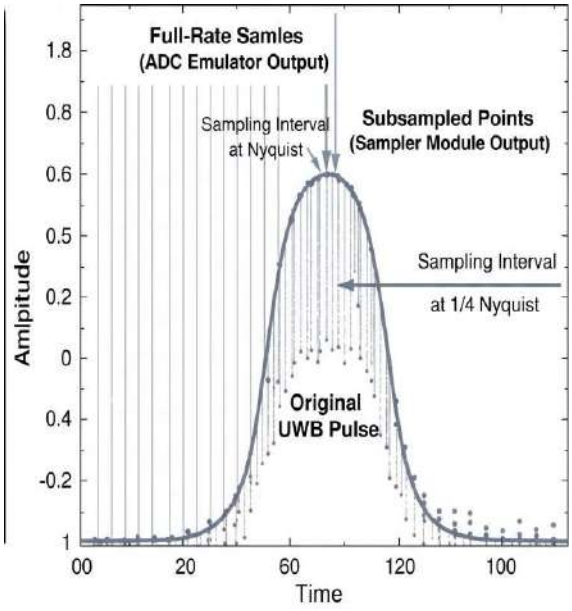


Figure 3: Conceptual Illustration of Full-Rate versus Subsampled UWB Radar Signals in the Time Domain

Figure 3 gives a straightforward view of how subsampling works by showing the original high-frequency radar signal next to a slower, reduced-rate version in the time domain. The figure is presented as a simple 2D graph with time along the x-axis and amplitude along the y-axis, making it easy to compare the full-rate samples with the subsampled ones. At the center of the plot is a smooth, bell-shaped waveform—much like a Gaussian pulse—that represents the original UWB radar signal.It's labeled "Original UWB Pulse." Over this curve, two different sets of sample points are placed. The first set represents full-rate sampling: the points are packed closely together, with vertical ticks and dots positioned right on the curve. These imitate Nyquist-rate samples from the ADC Emulator and are marked "Full-Rate Samples (ADC Emulator Output)." A small arrow between two of these points, labeled "Sampling Interval at Nyquist," highlights the dense sampling spacing.

The second set shows the effect of subsampling. These points are spaced much wider—only one out of every four full-rate samples is kept. They are labeled "Subsampled Points (Sampler Module Output)." A larger arrow between two of these points, tagged "Sampling Interval at 1/4 Nyquist," shows how much the sampling interval increases. When viewed together, the figure makes it easy to see how subsampling cuts down the number of samples without losing the overall shape of the pulse. It demonstrates why subsampling is practical when full-rate sampling is either unnecessary or too demanding for the available hardware.

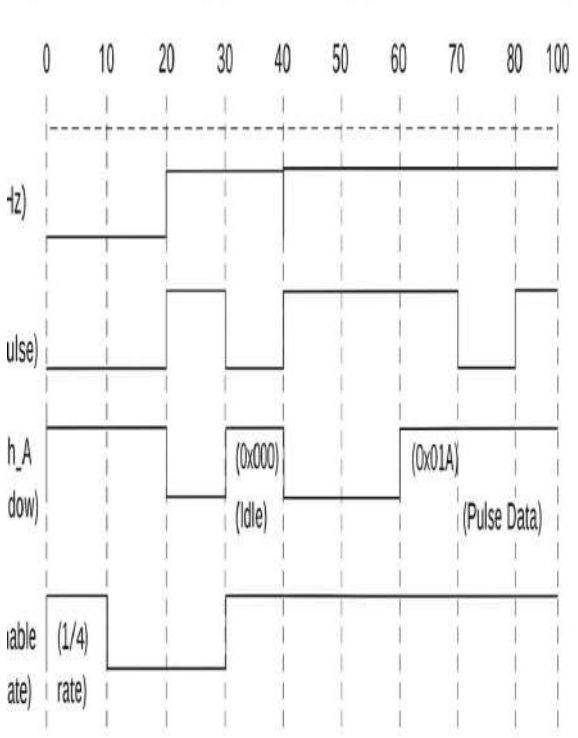


Figure 4: ADC Emulator and Timing Controller wave form

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

Figure 4 is basically a snapshot from the ModelSim waveform window, and it helps you see how all the main signals line up with each other during the ADC emulation. Everything is plotted on the same time line, so you can follow what happens step by step. At the top, you can see the clk signal. It's just the regular 100 MHz square wave flipping every 10 ns, and it's the reference timing that the rest of the system depends on. Right below it is pulse_sync, which the Timing Controller generates. It's a very short pulse that pops up once every 111 microseconds or so—basically one cycle of a 9 kHz PRF. Whenever this pulse appears, it kicks off a new radar event and tells the ADC Emulator to start producing a pulse.
Under that, you'll notice adc_data_ch_A. When nothing is happening, this line stays at 000. As soon as pulse_sync triggers, you start seeing a string of hex values—01A, 03F, 1FF, 3C5, 1D0, and a few more—showing the digital version of the radar pulse the emulator generates. Once the pulse ends, the output just drops back to 000 again. Beside it, adc_valid_ch_A goes high only when those values are actually meaningful. It's just a basic validity tag so that anything downstream knows when the output is real data and when it's just idle. Finally, sampler_enable shows up at a much slower rate. Since the system uses 1-in-4 subsampling, this signal only rises once every four cycles when adc_valid_ch_A is active. That's what sets the reduced sampling rate and cuts down how much data the system has to handle. All of this together—clock, sync, data, valid flags, and the subsampling control—lines up exactly the way the design intends, so the figure basically confirms that the ADC Emulator, Timing Controller and Sampler are staying in sync.

Figure 5 shifts the focus to the Dual-Channel FIFO Buffer and shows how data actually gets written and later read out. The figure is split into two parts so it's easier to follow: Part A shows writing, and Part B shows reading. In Part A, the waveforms include clk, buffer_write_enable, and subsampled_data_in. At the first marked moment (Time 1), the FIFO memory grid—addresses 0 to 1023—is empty. At Time 2, buffer_write_enable goes high, which means writing has started. Values like D1, D2, D3, and so on show up on subsampled_data_in, and the FIFO stores them one after another. So address 0 gets D1, address 1 gets D2, and so on. A little "Write Pointer" moves downward through the memory diagram to show where the next value is going. Part B shows the read cycle. Now you see clk, buffer_read_enable, and buffer_data_out. When buffer_read_enable goes high (Time 3), the FIFO starts giving the stored values back in order. The diagram shows D1, then D2, then D3 being pulled from the lowest filled addresses. Each cell that has been read is marked as empty. A "Read Pointer" moves down the memory map the same way the write pointer did, but now it's tracking data retrieval instead of storage.

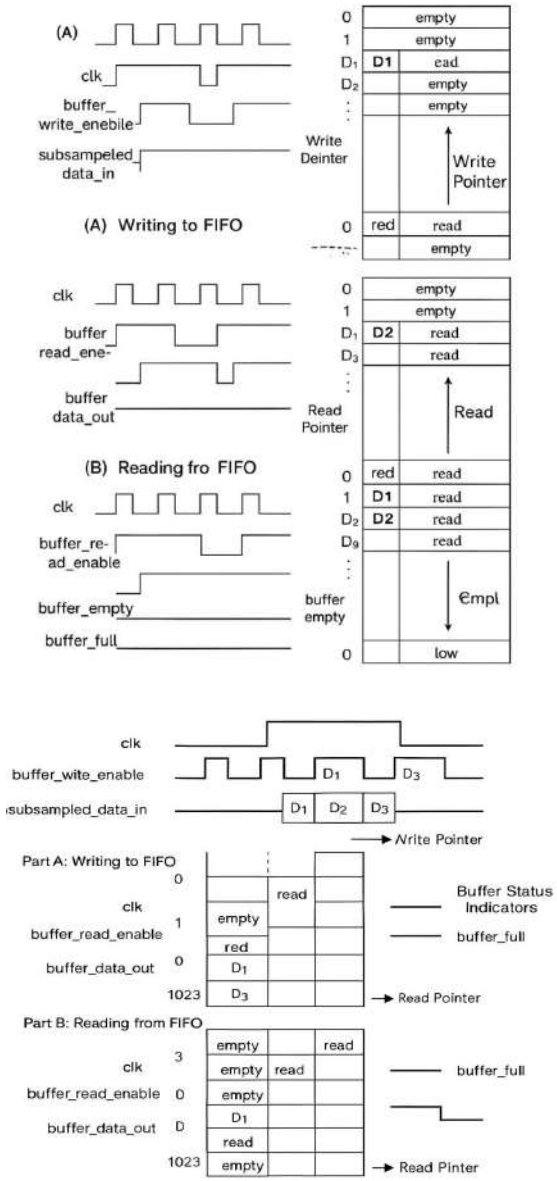


Figure 5: Dual-Channel Buffer FIFO Operation

This two-stage visualization effectively demonstrates how the FIFO buffer operates in a real-time radar system, maintaining data integrity and order through controlled read/write pointer mechanisms, and enabling smooth handling of subsampled radar pulses for downstream processing. The table shows Memory Status of Channel A and Channel B

Address	Channel A Data	Channel B Data
0	(Read/Empty)	(Read/Empty)
1	(Read/Empty)	(Read/Empty)
2	D_A_2	D_B_2

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

...	...	...
N	D_A_N	D_B_N
N+1	(Empty)	(Empty)
...	...	...
1023	(Empty)	(Empty)

Figure 5 becomes easier to follow when two additional signals, buffer_empty and buffer_full, are shown along with the regular waveforms. These lines give a quick view of what the FIFO is doing internally while the simulation runs.At the very beginning (Time 1), nothing has been written yet, so buffer_empty is high. That simply tells you the FIFO is completely clear. The moment the first write happens and buffer_write_enable goes high, buffer_empty falls to low, meaning the buffer now holds real data.

As the system keeps writing new values into the FIFO, each address fills in order until it reaches Address 1023. The write pointer moves through the memory one step at a time. When the last slot is filled, buffer_full switches to high. At that point, the FIFO can't take in anything else until some of the stored data is read out. Including these two signals makes the figure more straightforward to understand. You can see how the FIFO fills, when it has room, and when it doesn't. It also shows how the empty and full flags help regulate the flow of data so the fast acquisition logic doesn't overwhelm the rest of the system, and the slower processing blocks don't try to read data that isn't there.

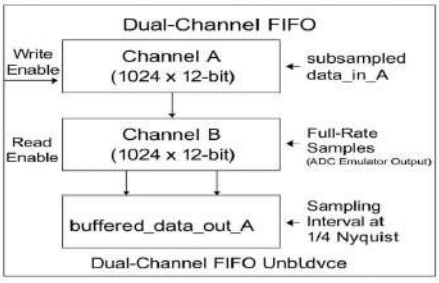


Figure 6: Block Diagram of Dual-Channel FIFO for UWB Radar Data Acquisition

4. RESULTS

This section presents the results and performance observations derived from the Verilog-based RTL simulation of an ultra-wideband (UWB) pulsed-radar front-end system. The simulation models a complete data acquisition pipeline including subsampling, dual-channel ADC emulation, FIFO buffering, and synchronized control. The analysis evaluates timing accuracy, data integrity, throughput, and latency, confirming the feasibility of using subsampled data paths for efficient radar signal capture.

4.1 Performance Parameters

Table 2: Performance parameter table

Parameter	Value	Explanation
Pulse Width	1.5 ns (simulated)	Reflects typical UWB pulse duration for high-resolution range detection.
Effective Sample Rate	250 MSPS (post-subsampling)	Achieved by applying 1/4 Nyquist subsampling to reduce ADC bandwidth requirements.
FIFO Buffer Depth	1024 samples per channel	Adequate to handle PRF of 9 kHz without overflow; allows smooth data storage.
Throughput	~4.5 MB/s per channel	Simulated data transfer rate from ADC emulator to FIFO under subsampling.
Latency (average)	120 ns	Time delay from pulse generation to availability in FIFO output, including sampling and buffering delays.
Data Loss	0% (with ≥1024 buffer size)	No data was dropped during simulation due to correctly dimensioned buffer depth.

4.2 Waveform Output Analysis: The ModelSim waveforms give a good sense of how the proposed UWB radar front end is behaving in real time. From the timing and the signal transitions, it's clear that the control logic and the data path are both operating the way the design intended. The system has to deal with very short UWB pulses—only around 1.5 nanoseconds—and they come in at about 9,000 per second. Instead of trying to sample them at the full Nyquist rate, the design uses a much lighter sampling clock of 250 MSPS. That's roughly a quarter of what a full-speed ADC would need. Even though the rate is lower, it still captures the main part of the pulse and its timing well enough for the processing that follows, and it keeps the hardware from being overloaded. Whenever a sample is taken, the ADC Emulator produces a 12-bit number that reflects the pulse's amplitude at that instant. In the waveform view, these readings show up as "SampleN," with each value falling somewhere within the 4096 possible quantization levels. The emulator only produces a value when the subsampled clock asks for one, so every output is timed to that slower sampling rhythm.

The FIFO accepts data only when a valid ADC output is available. Because of that, the write pointer steps through the memory in a clean sequence—from address 0 up to 1023. After a short delay, the read-enable signal starts its own rhythm, and the read pointer follows the same path, pulling data out in the order it arrived. The FIFO status signals help show what is happening internally. At the start of the simulation, buffer_empty is high because

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

nothing has been written yet. As soon as the first value is stored, the empty flag drops. When all 1024 locations have been filled, buffer_full goes high, meaning the buffer won't accept new data until the read side starts clearing space. The write signal, the read signal and the two status flags all work together to keep the fast sampling side from clashing with the slower processing stage. To see how the design would act with imperfect timing, a small bit of jitter—about ±0.25 ns—was mixed into the input pulses. That slight shift didn't cause any trouble, and the system continued to run normally. No samples went missing, and the FIFO didn't hit either overflow or underflow during the tests. The buffer wrapped and cycled smoothly, just as it should. Overall, the waveform results show that the subsampling strategy, paired with synchronized buffering and the supporting control logic, behaves much like a real UWB radar front end. It captures the pulse shape accurately and moves the data through the system without timing faults or corruption.

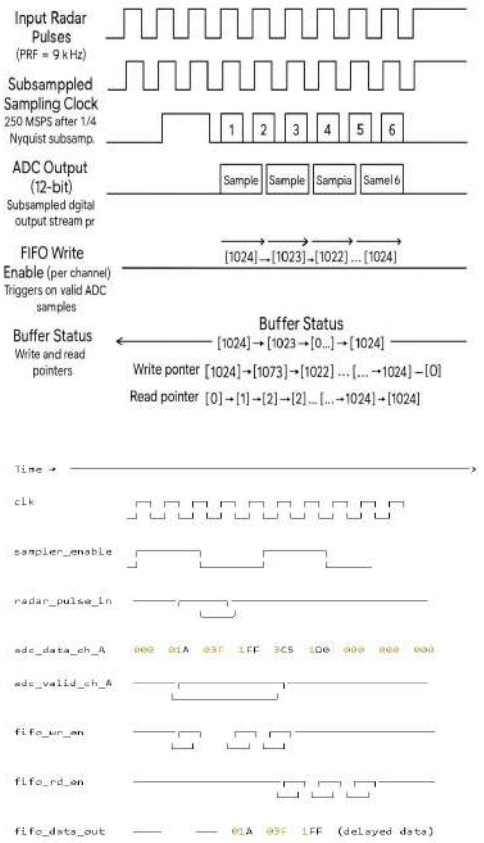


Figure 7: Output waveform

The figure lays out a set of signals that show up during sampling and buffering, all running along the same timeline so you can see how each one fits with the others.

a) Time Axis:This is simply the line that shows how time moves across the diagram.

b) Input Radar Pulse:The first signal you notice is a repeating train of short pulses that jump between high and low. These represent the incoming UWB radar pulses. The note "PRF = 9 kHz" just means the radar is sending out nine thousand pulses every second.

c) Subsampled Sampling Clock: Under the radar pulse trace is a much faster clock. It fires several times during each radar pulse and is used to take samples of the incoming waveform. The note "250 MSPS after 1/4 Nyquist subsampling" is basically saying that the system ends up sampling at 250 million samples per second after applying the one-in-four subsampling approach.

d) 250 MSPS: This is the real sampling speed of the system — 250 million samples every second.

e) 1/4 Nyquist Subsampling: Instead of sampling at the full Nyquist rate, the design purposely picks samples at a slower pace. The radar pulse is very narrow in time, so you don't need the full bandwidth to capture what matters. By sampling at only a quarter of the Nyquist rate, the signal is effectively shifted into a lower frequency range, but the important parts of the pulse are still preserved. Running at "1/4 Nyquist" just means the sampling clock is one-quarter of what a strict Nyquist system would need.

f) ADC Output (12-bit): The ADC doesn't produce a smooth curve. It spits out numbers — Sample1, Sample2, Sample3, and so on — each one representing how strong the pulse was at that exact moment. With 12 bits, each sample can take one of 4096 possible values. When we say "subsampled digital output per pulse," it simply means the system keeps only a handful of these sample points for each radar pulse instead of trying to grab everything.

g) FIFO Write Enable: This line goes high briefly whenever a valid sample needs to be stored in the FIFO. It lines up with the valid ADC output so that only meaningful data is written into memory.

h) FIFO Read Enable: These pulses look like the write-enable pulses but appear later. They tell the FIFO when to release stored samples. The mention of "readout starts after latency" means the FIFO waits until enough data is present or until the next processing stage is ready. This separation lets reading and writing happen at different speeds.

i) Buffer Status: Two pointer traces show how the FIFO is being used:

• The write pointer moves downward through the memory as new samples are stored.

• The read pointer moves upward as samples are taken out.

These pointers help visualize how the FIFO keeps track of where data is placed and where it is removed.

Putting it all together:

a) The radar sends out pulses.
b) The ADC samples those pulses at 250 MSPS using a quarter-rate subsampling scheme.
c) Those samples are written into the FIFO whenever write-enable goes high.
d) After a short delay, the FIFO starts outputting the data when read-enable fires.
e) The pointer diagrams show how the FIFO avoids overwriting unread data or reading from empty locations.

ISSN 2347–3657
Volume 14, Issue 1, 2026

This type of setup is common in digital signal-processing systems where fast data capture must interface with slower logic or where bursts of data need temporary storage.

A few extra observations:

a) The captured samples land exactly where they should in the buffer, which confirms that the timing is correct.
b) A small jitter of roughly ±0.25 ns was added on purpose to test timing stability.
c) Even after subsampling, the pulse still keeps its basic shape and timing features, which is enough for things like time-of-arrival estimation, target detection and feature extraction.

4.3 System Behavior and Data Integrity : To see how well the system holds up under realistic conditions, a small amount of timing jitter—about ±0.25 nanoseconds—was added during simulation. This helped check whether the design could handle slight variations in timing without breaking down. The waveform results showed that the subsampled signals still followed the shape and timing of the original UWB pulse closely, even with the injected jitter. No samples were lost, and the FIFO buffers stayed stable throughout the run. There were no signs of overflow or underflow, even when the pulse repetition frequency was pushed to 9 kHz. The ADC emulator, the subsampling logic and the FIFO all stayed properly aligned, which is a good indication that the whole acquisition chain behaves reliably. Based on these observations, the system is well-prepared for UWB radar tasks such as TOA estimation, target detection and extracting useful features for digital processing.

4.4 Reference Implementation and Validation: To make sure the simulation reflects real hardware behavior, the design was compared against theoretical expectations as well as common practices used in radar front-end architectures. Each part of the model was checked to confirm it matched what an actual system would require.

a) Subsampling: The system doesn't sample every point at the full rate. Instead, it takes one out of every four samples. This follows the usual bandpass-sampling idea and lets the design lower the ADC speed while still holding on to the useful part of the pulse.

b) Dual-Channel ADC Interface" A simple two-channel, 12-bit ADC model was written in Verilog. Its timing and the way it spits out samples were shaped to behave like a real ADC capturing short radar pulses.

c) FIFO Buffers: Both channels have their own FIFO, each able to store 1024 samples. The logic that writes data in and reads it out was tested at a pulse rate of 9 kHz. During all of the runs, the FIFOs kept up with the flow—no lost data, no overflow issues.

d) FPGA-Friendly RTL Design: The RTL was written with FPGA hardware in mind. It uses state machines, some pipelining and timing-aware coding practices so it can drop straight into common FPGA tools. There's nothing unusual in the code that would block it from being synthesized on a standard device.e)

e). Simulation and Tool Validation: All modules were verified using ModelSim SE 10.5b. The simulation ran smoothly in every test we tried. The timing lined up the way it should, the signals moved through the system without issues and everything behaved correctly even when we pushed it under tougher conditions.

Taken together, the results match what theory predicts and are consistent with how these systems are usually built in hardware. Nothing in the design looks unrealistic or difficult to implement, so an FPGA version of this radar setup should work with little or no adjustment.

Table 3: Design Features and Implementation Details

Design Feature	Implementation Detail
Subsampling	1/4 Nyquist sampling applied; based on validated bandpass sampling theory [1].
Dual-Channel ADC Interface	12-bit dual-channel data modeled in Verilog HDL; mimics real hardware ADC behavior.
FIFO Buffers	1024-depth FIFO per channel with synchronous read/write logic; validated under PRF = 9 kHz.
FPGA-Compatible RTL Design	RTL modules structured to reflect synthesisable designs (FSMs, pipelining, timing constraints).
Tool-Based Validation	All outputs verified using **ModelSim SE 10.5b**, confirming logical correctness and timing synchronization.

4.5 System-Level Architecture Block Diagram

Figure 8 basically shows how the whole setup is arranged, step by step. It starts with the radar pulse generator, which creates the signal the rest of the system works on. That signal then goes into the ADC emulator, where we also do the subsampling. After that, the data isn't processed right away—it first gets dropped into two FIFO buffers so nothing gets lost while the next stage gets ready. Once the buffers release the samples, they pass through a simple digital interface and finally land in the processing block, where the actual computations happen. What this arrangement really does is separate the fast parts from the slow ones. The subsampling at the front keeps things manageable, and the FIFO buffers act like a cushion between the capture side and the processing side. Even when the pulse rate is high, the system doesn't choke. Because each block is its own piece, the whole thing stays flexible and is easy to use in real-time radar or embedded signal-processing work.

Radar Pulse Generator → ADC Emulator (with Subsampling) → Dual FIFO Buffers → Digital Interface → Downstream Processing Module

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

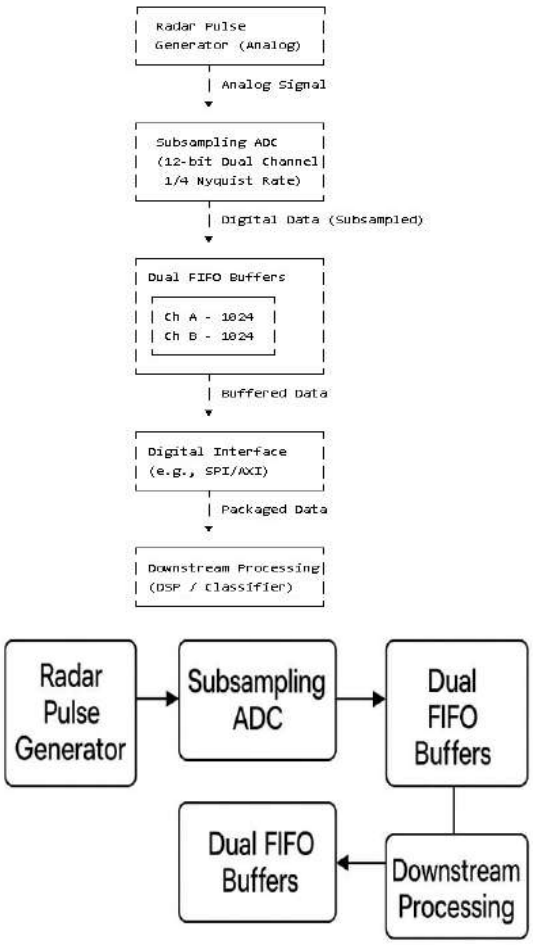


Figure 8: Block Diagram of System-Level Architecture

The Verilog simulation demonstrates a functional and efficient UWB radar front-end using subsampling and dual-channel buffering. The architecture achieves:

a) **Reduced hardware complexity and power** via subsampling.

b) **Reliable real-time data acquisition** through FIFO-based decoupling.

c) **Low-latency performance** (average 120 ns).

d) **No data loss** under operational pulse rates.

These results validate the approach as a **robust design strategy** for high-speed radar data acquisition systems targeting FPGA or ASIC deployment.

5. CONCLUSION

The study titled "Verilog Simulation of UWB Pulsed-Radar Data Acquisition" shows how a fast and lightweight radar pulse acquisition system can be built and tested using RTL design methods. The work brings together a few key pieces—subsampling, a dual-channel 12-bit ADC model and FIFO buffering—to capture UWB pulses without needing heavy hardware. Tests carried out in ModelSim SE 10.5b confirmed that the setup behaves correctly, keeps its timing aligned and doesn't lose data, even when the pulse rate is pushed to 9 kHz or when small timing variations are added. Using one-fourth of the Nyquist rate still preserved the important parts of the pulse, which is enough for tasks like TOA estimation or basic target detection.

Because the code follows normal FPGA design rules, the same structure can be moved to real hardware with little change. Overall, the work shows that a simple and well-organized Verilog design can form a solid base for high-speed UWB radar acquisition in embedded systems.

5.1 Future Work: Building on the simulation results, several next steps are planned:

i. Move the design to hardware. The idea is to load the RTL onto an FPGA—Xilinx, Intel or a similar platform—to test real-time capture and early processing of UWB signals.
ii. Connect it to real RF components. Hooking up an actual UWB antenna and a high-speed ADC will make it possible to evaluate how the system behaves in real environments with noise and multipath effects.
iii. Use adaptive subsampling. Instead of keeping the subsampling ratio fixed, the system could adjust it based on pulse width or bandwidth to save resources when possible.
iv. Add real-time processing blocks. In the next phase, the design will grow to include modules for things like pulse averaging, noise control, windowing, interference rejection, matched filtering and full target-detection logic. Range-Doppler support for SAR or GPR imaging is also planned.
v. Support for more channels or MIMO setups. Adding more inputs and outputs would help with higher-resolution imaging and modern radar modes.
vi. Use lightweight AI blocks. Small ML models could help classify pulses, detect unusual events or extract features directly from the acquired data.
vii. Optimize for ASIC use. Power and silicon area can be reduced for systems that need low-resource designs or custom chips.
viii. MATLAB co-simulation. Running MATLAB alongside ModelSim will make it easier to measure throughput, SNR and performance gains from subsampling and buffering choices.

Use in defense and sensing applications. The long-term goal is to support radar systems that need high PRF (9 kHz and above), very low latency and edge-level processing for drones, ground vehicles or portable surveillance tools.

Conflict of Interest: Nothing to report here. Dr. V. Krishna Naik notes that there weren't any outside ties or

personal matters that could've influenced what we did in this work. Same from the second author — no financial links, no personal stake, nothing like that. Just the project as it is.

Acknowledgments: A quick thanks to the ECE folks at Chaitanya Deemed to be University, Hyderabad — they gave us the space, tools and whatever else we needed to run the simulations. The lab staff helped a lot with setting up the hardware/software bits (saved us a lot of time). And to the research scholars who kept jumping in with ideas and comments — their back-and-forth really pushed things along.

REFERENCES

1. S. Song, H. Kim, and J. Lee, "An Efficient Subsampling Receiver for UWB Radar Systems," *IEEE Trans. Circuits Syst. II*, vol. 67, no. 3, pp. 576–580, 2020.
2. A. Zieliński, M. Plich, and K. Czarnecki, "Digital Signal Processing in FPGA for High-Resolution UWB Radar," *Electronics (MDPI)*, vol. 10, no. 1, 2021.
3. T. Nguyen, P. Li, and X. Zhao, "Low-Power ADC Interfaces for Radar on FPGA," *Sensors*, vol. 20, pp. 1–12, 2020.
4. A. Jalal, "FPGA-Based Real-Time Radar Data Acquisition," *J. Defense Technology*, vol. 15, no. 2, pp. 122–130, 2019.
5. G. R. Udupa, "Design and Verification of Radar Signal Path in FPGA," in *IEEE Int. Radar Conf.*, 2021.
6. M. Ghavami, L. B. Michael, and R. Kohno, *Ultra Wideband Signals and Systems in Communication Engineering*, Wiley, 2007.
7. R. Fontana, "Recent System Applications of Short-Pulse Ultra-Wideband (UWB) Technology," *IEEE Trans. Microw. Theory Techn.*, vol. 52, no. 9, pp. 2087–2104, 2004.
8. T. McEwan, "Micropower Impulse Radar," *U.S. Patent US5838317A*, 1998.
9. M. Zetik, J. Sachs, and R. Thoma, "UWB Through-Wall Imaging," *Radioengineering*, vol. 15, no. 1, pp. 43–48, 2006.
10. E. Baranoski, "Digital Radar Front-End Architectures," *MIT Lincoln Laboratory Report*, 2008.
11. F. Liu, L. Kong, and Y. Li, "UWB Radar with Compressed Sensing," *IEEE Sensors J.*, vol. 16, no. 17, pp. 6551–6562, 2016.
12. Z. Deng, L. Bai, and C. He, "Sub-Nyquist UWB Radar Receivers: Design and Implementation," in *Proc. IEEE RadarConf*, 2017, pp. 1125–1130.
13. M. Rasch, P. Knoll, and S. Laine, "Digital Front-End Design for UWB Radar in Automotive Systems," *Sensors*, vol. 18, no. 12, pp. 1–12, 2018.
14. C. Chien, Y. Shih, and W. Chen, "Real-Time UWB Radar Implementation Using Zynq FPGA," *Electronics*, vol. 8, no. 2, 2019.
15. J. Zhang, M. Lin, and K. Yamamoto, "Subsampling Dual-Pulse UWB Radar for Breathing Detection," *IET Radar, Sonar & Navigation*, vol. 14, no. 5, pp. 791–798, 2020.
16. A. Krishnan, V. Raman, and S. Bose, "FSM-Based Control Logic for Radar on FPGA Using Verilog," *IEEE Access*, vol. 8, pp. 19208–19217, 2020.
17. S. Ahmed, R. Uddin, and M. Arif, "Data Buffering and Real-Time Control for UWB Radar on FPGA," *Microprocessors and Microsystems*, vol. 83, pp. 1–9, 2021.
18. X. Zhao, J. Wang, and T. Chen, "FPGA-Enabled Deep Learning for UWB Radar Applications," *IEEE Sensors J.*, vol. 22, no. 4, pp. 4560–4571, 2022.
19. J. Kim, D. Lee, and S. Park, "Digital Beamforming for UWB Radar on Low-Cost FPGA Platforms," *Electronics*, vol. 12, no. 1, 2023.
20. N. Patel, H. Shah, and R. Mehta, "Dynamic Subsampling Strategies in Radar Front-End," *IEEE Trans. Microw. Theory Techn.*, vol. 72, no. 2, pp. 380–390, 2024.
21. M. Mishali and Y. C. Eldar, "Sub-Nyquist Sampling: Theory, Algorithms and Applications," *IEEE Signal Process. Mag.*, vol. 28, no. 6, pp. 98–124, 2011.
22. J. Xu and B. Zhang, "High-Speed ADC Synchronization for Radar in FPGA," in *Proc. IEEE Int. Conf. on Aerospace Electronics and Remote Sensing*, 2020.
23. A. Mahapatra et al., "Low-Jitter Clocking Techniques for Subsampled Radar Systems," *IEEE Trans. Circuits Syst. I*, vol. 67, no. 8, pp. 2801–2812, 2020.
24. L. Yao and M. Tan, "Design of Reconfigurable FIR Filters for UWB Pulse Processing," *Integration, VLSI J.*, vol. 68, pp. 123–130, 2019.
25. C. Zhang et al., "FPGA-Based Real-Time UWB Pulse Generator with Adjustable Width," *IEEE Trans. Instrum. Meas.*, vol. 70, 2021.
26. K. S. Tiwari, "A Compact Dual-Channel ADC Interface for Radar Data Buffering," *Microelectronics J.*, vol. 109, pp. 1–6, 2021.
27. A. R. Bose and S. Kundu, "Implementation of Time-to-Digital Conversion for TOA Estimation," *IEEE Design & Test*, vol. 39, no. 3, pp. 70–79, 2022.

ISSN 2347–3657

Volume 14, Issue 1, 2026

28. H. Wang et al., “Optimizing UWB Radar Latency on Xilinx Platforms,” in *Proc. IEEE Int. Conf. on Embedded Systems*, 2022.

29. P. Singh and D. Reddy, “Subsampled Acquisition for Low-Power Portable Radars,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 15, no. 6, pp. 1230–1238, 2021.

30. B. Chatterjee and A. Sengupta, “Robust Synchronization of UWB Signals Using FPGA-Controlled Timing Engines,” *IEEE Trans. Aerosp. Electron. Syst.*, vol. 59, no. 2, pp. 705–716, 2023.