



ARL-TR-9796 • SEP 2023



Advanced Radar Waveform Generator

by William Diehl and Neal Tesny

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.



Advanced Radar Waveform Generator

William Diehl and Neal Tesny
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

1. REPORT DATE		2. REPORT TYPE		3. DATES COVERED	
September 2023		Technical Report		START DATE January 2022	END DATE July 2023
4. TITLE AND SUBTITLE Advanced Radar Waveform Generator					
5a. CONTRACT NUMBER		5b. GRANT NUMBER		5c. PROGRAM ELEMENT NUMBER	
5d. PROJECT NUMBER		5e. TASK NUMBER		5f. WORK UNIT NUMBER	
6. AUTHOR(S) William Diehl and Neal Tesny					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) DEVCOM Army Research Laboratory ATTN: FCDD-RLA-LD Adelphi, MD 20783-1138				8. PERFORMING ORGANIZATION REPORT NUMBER ARL-TR-9796	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)		11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.					
13. SUPPLEMENTARY NOTES ORCID ID: William Diehl, 0000-0002-6293-5018					
14. ABSTRACT We introduce the Advanced Radar Waveform Generator (ARWG). The ARWG is part of the Software Defined Electronic Warfare Family of Systems. It is implemented in the commercial AMD-Xilinx Gen3 Radio Frequency System-on-Chip in the ZCU208 Evaluation board form-factor. It provides a flexible platform for generating user-defined radar pulses with complex features such as intrapulse modulations, near real-time custom waveform generation, pulse-to-pulse frequency agility, user-defined amplitude, frequency and phase modulation, and composite pulse group generation. It includes multiple user interfaces including serial command line interface and Python-based graphical user interface. The ARWG is connected to a supporting personal computer (PC) via a Gigabit Ethernet interface and can therefore interact with powerful applications residing in PCs or across networks. The basic design and major functions of the ARWG are discussed and example features are showcased.					
15. SUBJECT TERMS electromagnetic warfare, radar, software-defined radio, field-programmable gate array, FPGA, Radio Frequency System on Chip, Electromagnetic Spectrum Sciences					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	
a. REPORT UNCLASSIFIED	b. ABSTRACT UNCLASSIFIED	c. THIS PAGE UNCLASSIFIED	UU	60	
19a. NAME OF RESPONSIBLE PERSON William Diehl				19b. PHONE NUMBER (Include area code) (301) 394-0931	

STANDARD FORM 298 (REV. 5/2020)

Prescribed by ANSI Std. Z39.18

Contents

List of Figures	v
List of Tables	vi
1. Introduction	1
2. Design Overview	2
2.1 Digital Design	2
2.2 Clock Generation	4
2.3 Interrupts	6
3. Signal Generators	9
3.1 Frequency Modulation Generator (FMGEN)	9
3.2 Binary Phase Shift Keying Generator (BPSKGEN)	13
3.3 Custom Generator (Custom) or Variable Direct Digital Synthesizer (VARDDS)	15
3.4 Signal Selector	22
4. Signal Modulators	22
4.1 Phase Rotator	22
4.2 Amplitude Modulator	25
4.3 Noise Generator	25
5. Numerically Controlled Oscillator (NCO) Controller	26
6. Frequency Agility	27
7. Composite Pulse Sequence Generator	28
8. Gigabit Ethernet	31
9. User Interfaces	32

9.1	Command Line Interface (CLI)	32
9.2	Graphical User Interface (GUI)	33
10.	Conclusions and Future Work	37
11.	References	38
	Appendix A. Vivado IP Integrator Block Design	40
	Appendix B. Python Script to Generate Custom Waveform	44
	List of Symbols, Abbreviations, and Acronyms	48
	Distribution List	51

List of Figures

Fig. 1	SDEWFoS	1
Fig. 2	SDEWFoS reference architecture	2
Fig. 3	ARWG major components.....	3
Fig. 4	ARWG clock provision including ZCU208 CLK104 peripheral, RFDC sampling clocks, and AXI streaming clocks	5
Fig. 5	PL-to-PS interrupt connection scheme	7
Fig. 6	Lead and trail interrupt template	8
Fig. 7	LFM low-to-high (0 to 4 MHz)	10
Fig. 8	LFM high-to-low (4 to 0 MHz)	11
Fig. 9	LFM low-to-high (2 to 4 MHz)	11
Fig. 10	LFM high-to-low (4 to 2 MHz)	11
Fig. 11	Vivado simulation of FMGEN-generated low-to-high NLFM and LFM pulses.....	12
Fig. 12	Vivado simulation of FMGEN-generated high-to-low NLFM and LFM pulses.....	12
Fig. 13	LFM spectrogram with 30 us PW and 20 MHz FE	13
Fig. 14	NLFM spectrogram with 30 us PW and 20 MHz FE	13
Fig. 15	BPSK with 5-bit BC.....	14
Fig. 16	Spectrum of BPSK with 13-bit BC	15
Fig. 17	BPSK with 13-bit BC with code transitions depicted at bottom	15
Fig. 18	Connection scheme for VARDDS ¹¹	16
Fig. 19	Custom waveform 1	17
Fig. 20	Custom waveform 2	18
Fig. 21	Custom waveform 3	18
Fig. 22	Custom waveform 4.....	19
Fig. 23	Custom waveform 5	19
Fig. 24	Custom waveform 6.....	20
Fig. 25	Custom waveform 7	20
Fig. 26	Custom waveform 8.....	21
Fig. 27	Phase rotation based on 1024 discrete angles (000 through 3FF hex)	23
Fig. 28	0° phase rotation	24
Fig. 29	180° phase rotation	24

Fig. 30	90° phase rotation	24
Fig. 31	Pulse sequence used to verify phase rotator accuracy	25
Fig. 32	LFM pulses using frequency agility mode with four channels.....	28
Fig. 33	LFM pulses using frequency agility mode with 16 channels	28
Fig. 34	Scheme for defining and running composite pulse group sequences .	29
Fig. 35	Sample pulse group sequence created using composite pulse generation.....	30
Fig. 36	GigE connecting RFSoc and PC in client-server configuration.....	31
Fig. 37	Command line interface menu	33
Fig. 38	ARWG full GUI display	34
Fig. 39	Generate-and-lock FM waveform from GUI.....	35
Fig. 40	GUI operation to read all DAC carrier frequencies and display in text boxes	35
Fig. 41	Generate-and-lock custom waveform using GUI	36
Fig. 42	Composite pulse group generation using the GUI.....	37
Fig. A-1	Advanced Radar Waveform Generator (ARWG) full block design...	41
Fig. A-2	Zoom on channel 1 portion of block design	42
Fig. A-3	Radio Frequency Digital Converter (RFDC), numerically controlled oscillator (NCO), timer, and interrupt concatenation portion of block design	43

List of Tables

Table 1	Software-Defined Electronic Warfare Family of Systems (SDEWFoS)	1
Table 2	Implementation statistics for ARWG in Xilinx Gen3 RFSoc, including major parameters.....	6
Table 3	Summary of ARWG interrupts including name, PL-PS shared interrupt number, and description	8
Table 4	Summary of FMGEN user-defined parameters with example definitions including CLI command sequence	9
Table 5	Implementation comparison of LFMGEN ¹¹ and FMGEN in Gen1 Xilinx RFSoc.....	10
Table 6	Summary of FMGEN user-defined parameters with example definitions including CLI command sequence	14
Table 7	Custom pulsed waveforms with FM defined according to approximate formula.....	17

Table 8	Summary of custom generator user-defined parameters with example definitions including CLI command sequence	21
Table 9	Summary of signal selector commands and effects, including CLI example sequences.....	22
Table 10	Summary of phase rotator commands and effects, including CLI example sequences.....	23
Table 11	Summary of amplitude modulator commands and effects, including CLI example sequences	25
Table 12	Summary of amplitude modulator commands and effects, including CLI example sequences	26
Table 13	NCO real-time interface ports (adapted from Xilinx ³)	26
Table 14	Summary of NCO Controller commands and effects, including CLI example sequences.....	27
Table 15	Summary of frequency agility commands and effects, including CLI example sequences.....	28
Table 16	CLI command sequence for aligning and starting the composite pulse group generator	30
Table 17	GigE Applications layer (custom communications interface) protocol	31
Table 18	Color coding of major groupings of GUI buttons.....	34

1. Introduction

The Advanced Radar Waveform Generator (ARWG) is part of the Software Defined Electronic Warfare Family of Systems (SDEWFoS). Software-defined radios (SDRs) such as the AMD-Xilinx Radio Frequency (RF) System on Chip (RFSoc) facilitate a flexible and low-cost environment for innovative RF techniques. The goal of SDEWFoS is to provide a landing platform for investigation of novel radar and electronic warfare (EW) techniques. The platform is based on the commercial off-the-shelf AMD-Xilinx Gen1/3 RFSoc and incorporates mostly commonly available software drivers and practices yet allows for easy insertion of custom software drivers and hardware intellectual property (IP) using standard software (SW)–hardware (HW) codesign practices.

The ARWG is one of four SDEWFoS “personalities.” The CAMO, Radar Warning Receiver (RWR), and Direct Radio Frequency Memory (DRFM) personalities are beyond the scope of this report. SDEWFoS personalities are summarized in Table 1 and Fig. 1.

Table 1 Software-Defined Electronic Warfare Family of Systems (SDEWFoS)

ARWG	Advanced Radar Waveform Generator
CAMO	Radio Frequency Emulator
RWR	Radar Warning Receiver
DRFM	Direct Radio Frequency Memory

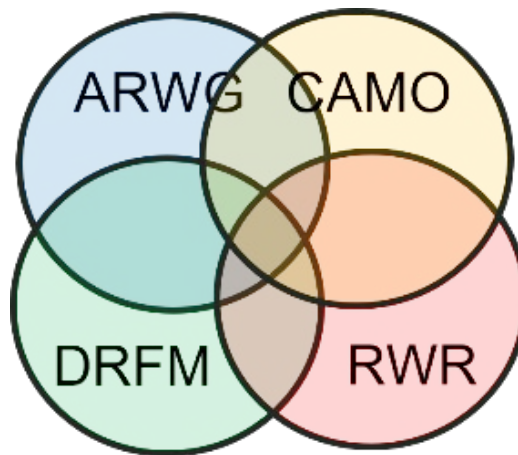


Fig. 1 SDEWFoS

SDEWFoS personalities share a common reference architecture where most of the design is common among all applications. The architecture shown in Fig. 2 is delineated into HW and SW layers, demarcated by the HW–SW interface.

Hardware components consisting of Xilinx, third party, or custom IP are centered around the Zynq processing system (PS)^{1,2} and the Xilinx RF Digital Converter (RFDC).³ The SW layers are the driver layer, transmit/receive layer, and mission layer. The driver layer contains primitive HW abstractions and consists mostly of Xilinx standard drivers.⁴ The transmit/receive layer contains higher-level drivers to control RF transmit, receive, and movement functions; this consists of a mix of Xilinx and custom drivers. ARWG does not have a receive function, so it uses only a transmit layer. The top layer is the mission layer, which contains custom-developed SW specific to the personality.

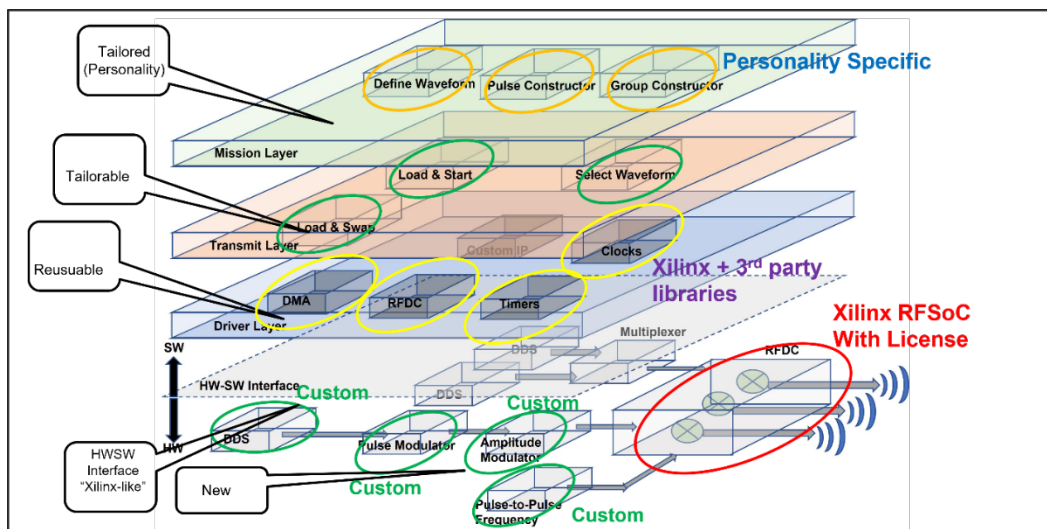


Fig. 2 SDEWfOS reference architecture

2. Design Overview

2.1 Digital Design

The ARWG is implemented in a baremetal configuration on the Xilinx Gen3 RFSoc in the ZCU208 evaluation board form factor. This includes the Advanced Reduced Instruction Set Computer Machine (ARM) Cortex A53 quad core^{*} microprocessor in the Zynq PS, and the XCZU48DR-2FSVG1517 Ultrascale+ field-programmable gate array (FPGA) in the programmable logic (PL) partition.¹ This facilitates powerful SW–HW codesign, where control-centric and latency-tolerant processes are implemented using the flexible C programming language in the PS, and data-centric and low-latency components are implemented in the PL. Industry standard interfaces, including Advanced Extensible Interface (AXI)^{4,5} and Xilinx SW drivers,⁴ are employed. Custom SW is written in standard C, including

^{*}Only one core is used in this baremetal implementation.

applications and hardware abstraction layer (HAL) drivers, and follows conventions of Xilinx embedded drivers⁴ to the extent possible. All IP modules used in the PL in this design are custom designed, are written in Very High-Speed Integrated Circuit Hardware Description Language (HDL/VHDL) and use register transfer level (RTL) design methodology; however, future expansions could easily incorporate IP produced by any method, for example, Vitis HLS or MathWorks HDL Coder.

The major design components are shown in Fig. 3. The key signal generation component is the “Channel,” consisting of four generators (Frequency Modulation Generator [FMGEN], Binary Phase Shift Keyed Generator [BPSKGEN], Custom 1, and Custom 2), signal selector, and signal modulators (phase, amplitude, and noise). A finalized ARWG design will include four channels; this intermediate design has only one channel. The memory access components include the Scatter–Gather Direct Memory Access (SG–DMA) peripherals and custom generator selector. The RFDC is a Xilinx Gen3 IP and converts digital signals to RF. Four custom numerically controlled oscillator (NCO) controller IP instances can dynamically adjust RFDC oscillator carrier frequencies (CFs) on each digital-to-analog (DAC) tile independently. The NCO controllers are low latency and can adjust CF in less than 2 μ s upon command from SW. Finally, four Xilinx AXI timers are included in this design; the timers perform time-sensitive functions such as complex pulse generation and diagnostic functions. The actual RF output is facilitated by the XM655 daughter card,⁵ which provides SMA balanced–unbalanced (balun) RF connections for downstream RF consumption. The full ARWG Vivado IP Integrator Block Design is in Appendix A.

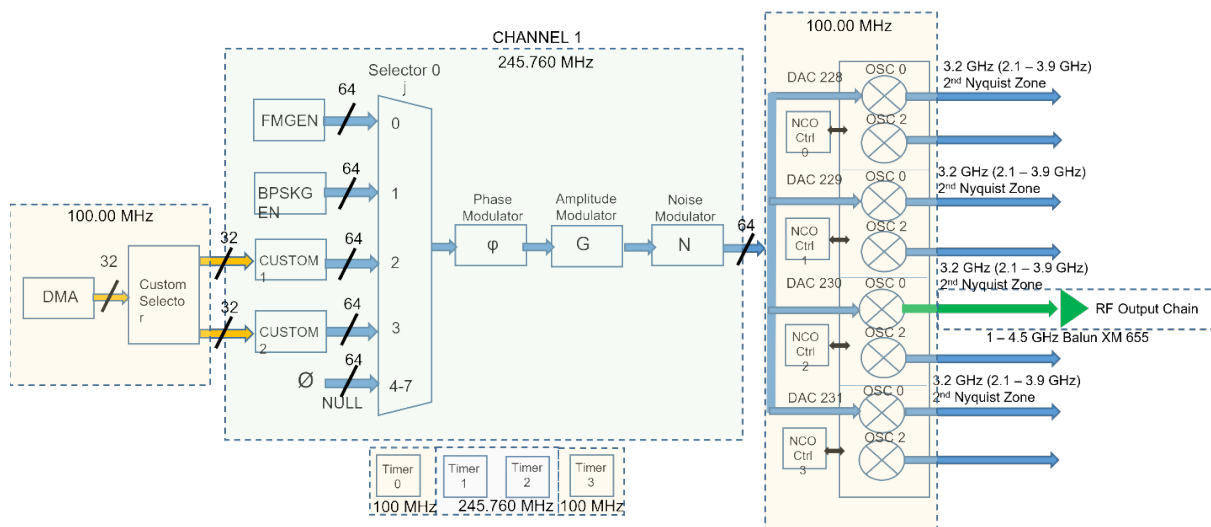


Fig. 3 ARWG major components

This design uses 31,676 Combinational Logic Block (CLB) look-up tables (LUT), 7,118 CLB slices, 157 Block Random Access Memory (BRAM) tiles, and 17 DSP48E2 Digital Signal Processor (DSP) tiles. These represent 7.45%, 13.39%, 14.54%, and 0.40% of LUTs, slices, BRAM tiles, and DSPs, respectively, meaning there is plenty of room for additional logic. The low DSP usage is because the ARWG is a signal generator, not a signal “processor;” other SDEWFoS components such as RWR and DRFM have higher DSP usage. This design uses a total of 8.020 W on-chip power (including static and dynamic), the largest power consumers of which are the RFDC at 3.881 W, the PS at 2.259 W, BRAM at 0.195 W, and clocks at 0.161 W. CLB power usage (particularly dynamic power) is relatively low.

2.2 Clock Generation

The ARWG was first implemented on the Gen1 RFSoc evaluation board ZCU111.⁷ However, early in the design cycle the decision was made to transition ARWG and future SDEWFoS personalities to the more-capable ZCU208. An important first step in this upgrade is to conform to clock generation on the ZCU208. The ZCU208 uses the CLK104 peripheral,⁸ which includes Texas Instruments LMK 04828 and LMX 2594 phase-locked loops (PLLs). The excitation and control mechanisms are different from the ZCU111, including serial peripheral interface (SPI) in lieu of inter-integrated circuits that necessitate the installation of a general purpose input–output (I/O) IP in the PL capable of driving SPI and new software drivers. References 8–10 were very helpful in implementing this change.

The LMK 04828 PLL is operated as a jitter cleaner with an output frequency of 245.76 MHz, and the cascaded LMX 2594 is operated as the PLL reference clock with a frequency of 491.52 MHz. This input comes onto the RFSoc chip as a differential-input clock I/O buffer, and directly drives the RFDC internal PLL with the 491.52-MHz reference signal. This RFDC PLL is located in Tile 230; Tile 230 PLL is used to drive external clock signals on Tiles 228, 229, 230, and 231. This PLL generates an RF-DAC sampling frequency of 3.93216 GSps and an AXI Stream frequency of 245.760 used for all signal processing IP in the PL. Onboard AXI Stream clock reference is generated using the Xilinx Clock Wizard IP. The PL control domain is sourced from the PL Clock in the PS at 100.0 MHz. Note that most Xilinx IP peripherals, including DMA and RFDC, must be sourced by this lower-speed clock. Additionally, any control inputs to the NCO bus on the RFDC must be at PL clock speed.

Choosing compatible frequencies for PLLs, RFDC sampling clocks, and AXI Stream is not trivial, as components often come from different manufacturers. The

Python PLL adjustment script¹¹ is combined with Texas Instruments TICS Pro software to choose these frequencies, although the Python script is updated to account for different operating parameters of the voltage-controlled oscillator (VCO) in the Gen3 RFDC. Further, the AXI Stream frequency of 245.760 was only barely attained in timing closure, as worst negative slack (WNS) on this clock network is 0.084 ns. This means that the AXI Stream frequency will likely have to be adjusted downward as designs get larger, and that flexible clock planning is an enduring requirement.

The clock distribution overview is shown in Fig. 4. The primary two clock domains—AXI Streaming domain and PL control domain—are shown in Fig. 3 and are annotated with their respective frequencies.

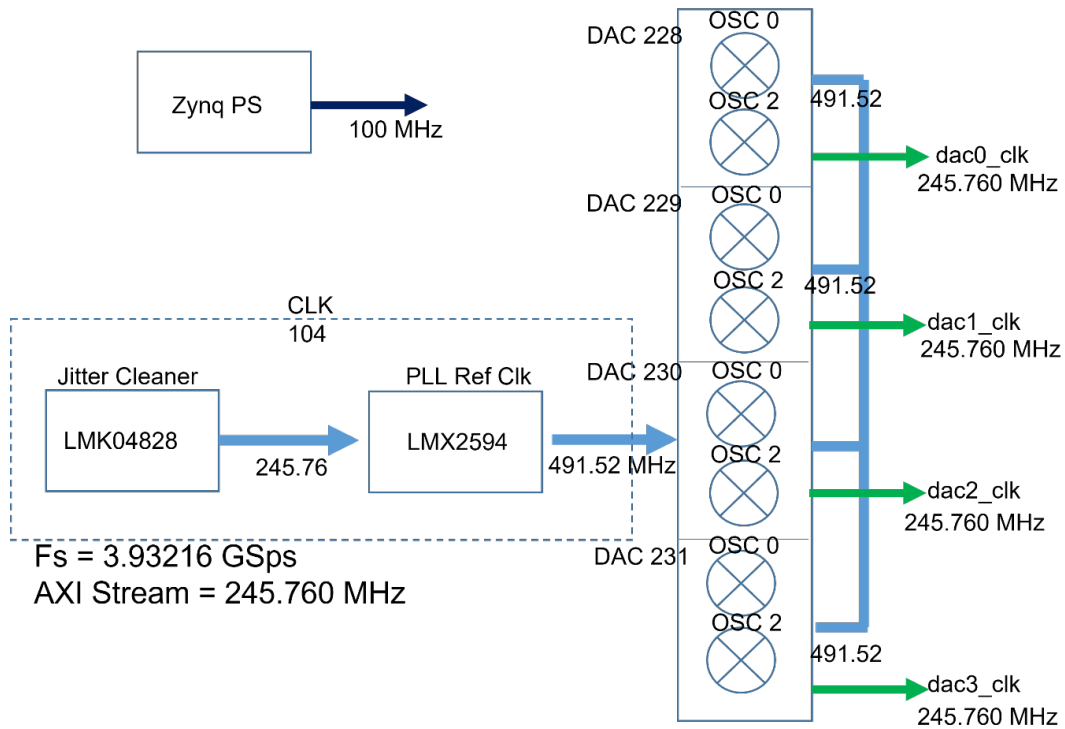


Fig. 4 ARWG clock provision including ZCU208 CLK104 peripheral, RFDC sampling clocks, and AXI streaming clocks

Of note, in a final design consisting of four channels (one serving each DAC Tile), each channel will operate in its own clock domain at the same frequency. In this design, however, there is only one channel, hence only one AXI Streaming domain.

The design parameters are summarized in Table 2.

Table 2 Implementation statistics for ARWG in Xilinx Gen3 RFSoc, including major parameters

Clocks			
	This design	Max	
	MHz	MHz	
RFDAC Fs	3932.16	7000	
LMK04828	245.76		
LMX2594	491.52		
RFDC PLL Ref	491.52		
AXI Stream	245.76		
PL Clock	100.00		
Utilization			
Component	This design	Available	Percentage
CLB LUT	31676	425280	7.45%
CLB Registers	21942	*	*
CLB Slice	7118	53160	13.39%
BRAM Tiles	157	1080	14.54%
BRAM32	149	*	*
BRAM18	16	*	*
DSP48E2	17	4272	0.40%
Timing			
	ns		
WNS	0.084		
Power			
	W		
Total	8.020		
RFDC	3.881		
PS8	2.259		
BRAM	0.195		
Clocks	0.161		

2.3 Interrupts

A robust system of interrupts is required for low-latency hardware applications to communicate status to the PS. In this design, the generic interrupt controller (SCUGIC) of the ARM Core A53 is used to process interrupts.² The 16 PL-to-PS shared interrupts are mapped to SCUGIC in accordance with Fig. 5 and Table 3. Interrupts are used for signaling from all signal generators (FMGEN, BPSKGEN, Custom 1, and Custom 2), custom generator selector, NCO controllers, DMA, and AXI timers. In particular, each signal generator provides at least two interrupts: the “lead” (pulse leading edge) and “trail” (pulse trailing edge). Knowing when a pulse has started and when it has finished greatly facilitates implementation of complex pulse generation protocols, including frequency agility and composite pulse groups. The lead and trail interrupt template is shown in Fig. 6. This interrupt

implementation does not easily scale to multiple channels, that is, the number of interrupts required will scale linearly with the number of channels. Therefore, future expansions should use a more robust interrupt handling architecture, probably assisted by a Xilinx Interrupt Controller IP.

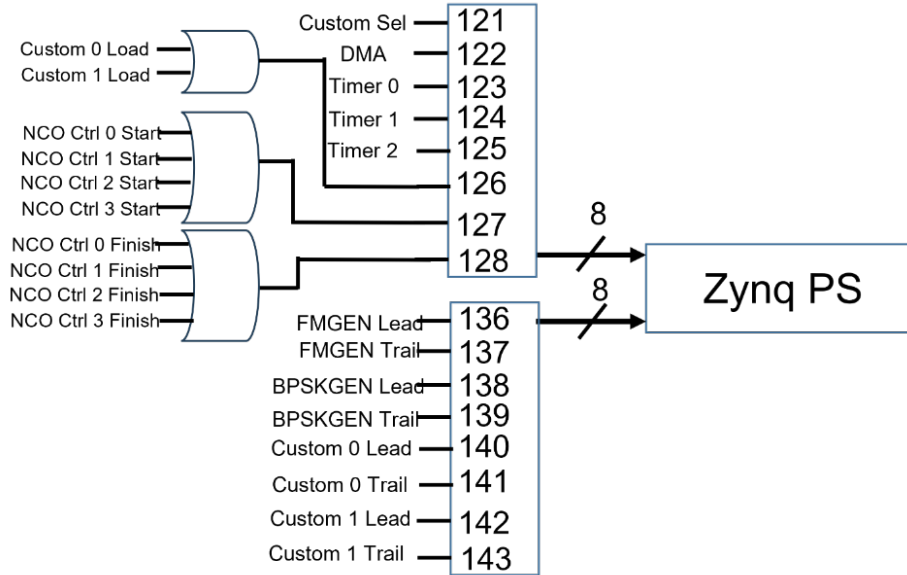


Fig. 5 PL-to-PS interrupt connection scheme

Table 3 Summary of ARWG interrupts including name, PL–PS shared interrupt number, and description

Name	No.	Description
Custom (VARDDS) Selector	121	AXI Stream Protocol locked to selected Custom (VARDDS) Generator
DMA (MM2S)	122	DMA Read (Memory Map to Stream) transaction complete
AXI Timer 0	123	Timer reaches event (e.g., countdown to 0)
AXI Timer 1	124	Timer reaches event (e.g., countdown to 0)
AXI Timer 2	125	Timer reaches event (e.g., countdown to 0)
Custom (VARDDS) 0 Load	126	Custom (VARDDS) Generator completes load from DMA
Custom (VARDDS) 1 Load	126	Custom (VARDDS) Generator completes load from DMA
NCO Controller 0 Start	127	NCO Controller starts update event
NCO Controller 1 Start	127	NCO Controller starts update event
NCO Controller 2 Start	127	NCO Controller starts update event
NCO Controller 3 Start	127	NCO Controller starts update event
NCO Controller 0 Finish	128	NCO Controller finishes update event
NCO Controller 1 Finish	128	NCO Controller finishes update event
NCO Controller 2 Finish	128	NCO Controller finishes update event
NCO Controller 3 Finish	128	NCO Controller finishes update event
FMGEN Lead	136	FM Generator Pulse Leading Edge
FMGEN Trail	137	FM Generator Pulse Trailing Edge
BPSKGEN Lead	138	BPSK Generator Pulse Leading Edge
BPSKGEN Trail	139	BPSK Generator Pulse Trailing Edge
Custom (VARDDS) 0 Lead	140	Custom (VARDDS) Generator Pulse Leading Edge
Custom (VARDDS) 0 Trail	141	Custom (VARDDS) Generator Pulse Trailing Edge
Custom (VARDDS) 1 Lead	142	Custom (VARDDS) Generator Pulse Leading Edge
Custom (VARDDS) 1 Trail	143	Custom (VARDDS) Generator Pulse Trailing Edge

Note: VARDDS = variable direct digital synthesizer; MM2S = memory map to stream.

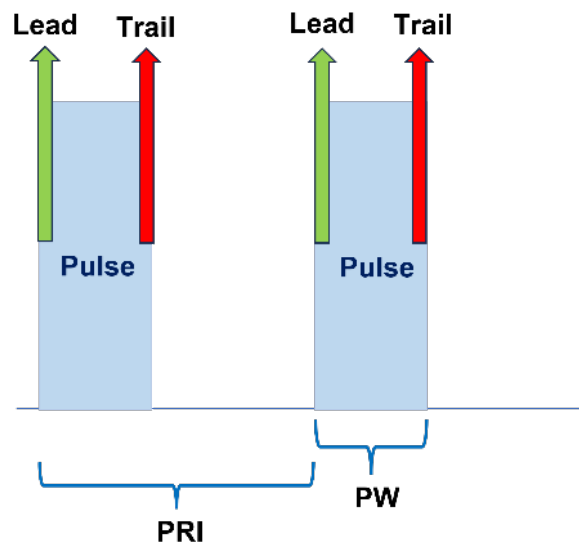


Fig. 6 Lead and trail interrupt template

3. Signal Generators

3.1 Frequency Modulation Generator (FMGEN)

The FMGEN is capable of synthesizing FM modulated pulses. The pulses are user-defined and include the following parameters: pulse width (PW), pulse repetition interval (PRI), frequency excursion (i.e., bandwidth) (FE), start frequency (Fstart), direction (Dir) low to high (L=>H) or high to low (H=>L), and linearity (linear or nonlinear FM [LFM or NLFM]). Table 4 summarizes the FMGEN parameters and the CLI command sequences (with examples) necessary to input and execute a new FM waveform. Some parameters are implicit; for example, end frequency (Fend) is automatically computed based on Fstart, Dir, and FE. The number of FM waveforms that can be defined in ARWG is arbitrary and is set as an array-of-structures bound at compile time.

Table 4 Summary of FMGEN user-defined parameters with example definitions including CLI command sequence

Parameter and command	Abbr/unit	Example	Effect
Pulse width	PW (ns)	s fa0 <cr>	PW = 4000 ns
Pulse repetition interval	PRI (ns)	t 3e80 <cr>	PRI = 16000 ns
Frequency excursion	FE (kHz)	u fa0 <cr>	FE = 4000 kHz
New start frequency	Fstart (kHz)	U fa0 <cr>	Fstart = 4000 kHz (Default = 0)
Direction (0=L=>H; 1=H=>L)	Dir	W 0 <cr>	Dir = L=>H (Default = L=>H)
Linearity (0=LFM; 1=NLFM)	Lin	y 0 <cr>	Lin = LFM (Default = LFM)
Lock FM waveform	...	v 0 <cr>	Parameters locked to FM waveform 0
Load and run FM waveform	...	l 0 <cr>	FM waveform 0 executed on FMGEN

The FMGEN in the ARWG is based on the linear frequency modulation generator (LFMGEN) detailed in Diehl and Viveiros¹¹ but includes significant upgrades. Upgrades include ability to specify L=>H or H=>L directions, arbitrary start (and end) frequencies, and provision of a nonlinear option. NLFM on pulse waveforms are used in many current and projected radars. Pulse compression with NLFM does not require amplitude weighting for the suppression of time-sidelobes generated by LFM compression, resulting in better signal-to-noise ratio with less signal processing complexity.¹²

The LFMGEN IP in Diehl and Viveiros¹¹ was upgraded with new capability and renamed FMGEN. A comparison between LFMGEN IP¹¹ and the new FMGEN is shown in Table 5. Only a 1% increase in FPGA resources was required to implement the new capability; most of the resources in FMGEN consist of the static reference sinusoidal accesses as distributed RAM LUTs. The LFMGEN

C-language drivers are updated but are otherwise fully backward compatible with previous applications such as Diehl and Viveiros.¹¹

Table 5 Implementation comparison of LFMGEN¹¹ and FMGEN in Gen1 Xilinx RFSoc

	LFMGEN	FMGEN	Delta
LUT	7179	7259	1.1%
BRAM	0	0	0
DSP	0	1	1
Static (W)	1.088	1.088	0.0%
Dynamic (W)	0.409	0.419	2.4%
WNS (ns)	0.928	0.548	-40.9%

In Table 5, both generators are implemented with two in-phase and quadrature (I&Q) samples (2I, 2Q) per clock cycle, a 250 MHz stream clock frequency (500 MSPS), and 3.95 GHz maximum frequency in the second Nyquist zone. LUT increases denote that only minor increased resources for significant increase in capability. A WNS of 0.548 ns indicates that this IP can probably run at a much higher clock frequency.

Sample LFM pulse sequences generated with FMGEN are shown in Figs. 7–10.

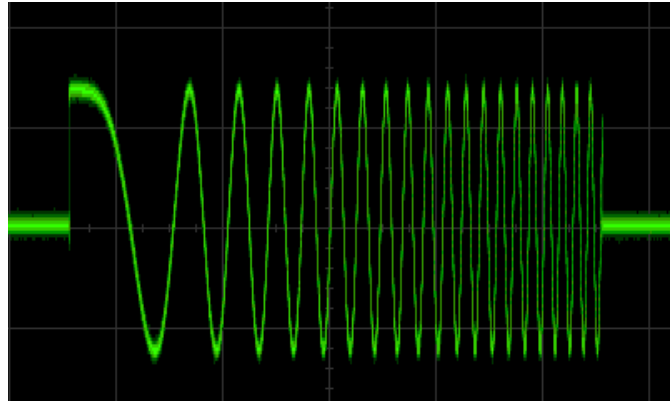


Fig. 7 LFM low-to-high (0 to 4 MHz)



Fig. 8 LFM high-to-low (4 to 0 MHz)

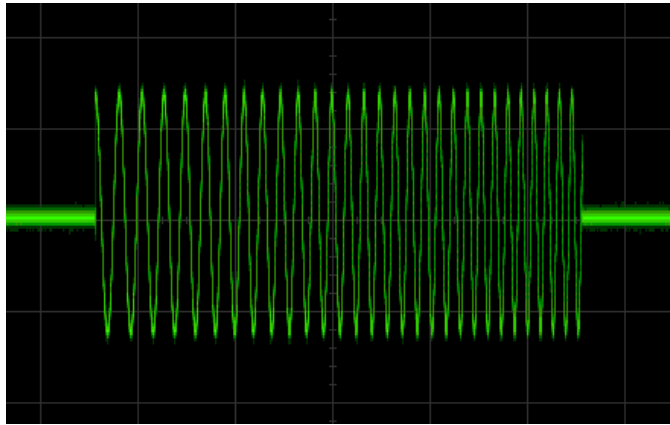


Fig. 9 LFM low-to-high (2 to 4 MHz)

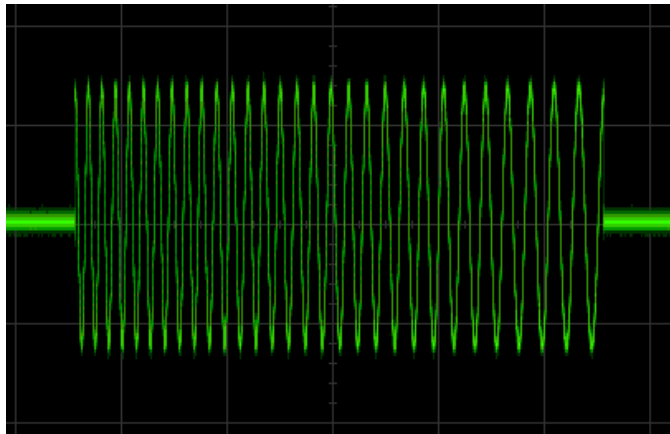


Fig. 10 LFM high-to-low (4 to 2 MHz)

The differences between LFM and NLFM are best visualized not in the time domain but in the time-frequency domain in a spectrogram. Pulses in Figs. 11 and 12 are generated using FMGEN in Vivado simulator. Figures 13 and 14 are real RF signals

at 3.2 GHz CF; the pulse in Fig. 13 is linear, and the pulse in Fig. 14 is nonlinear, where the $f(t)$ is proportional to t^2 , or $df(t)/dt$ is linearly proportional to t . Other NLFM implementations can easily be implemented in FMGEN; more complex nonlinear waveforms are best coded using the custom generators to be described in Section 3.3.

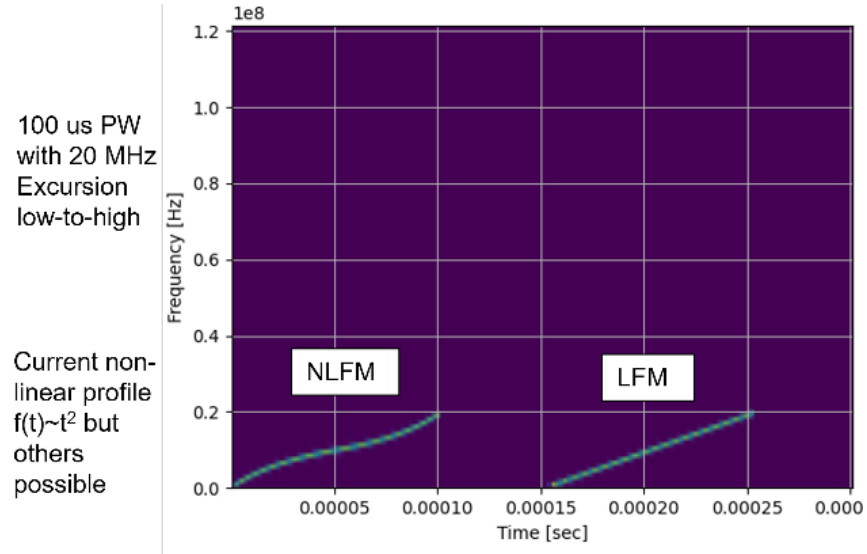


Fig. 11 Vivado simulation of FMGEN-generated low-to-high NLFM and LFM pulses

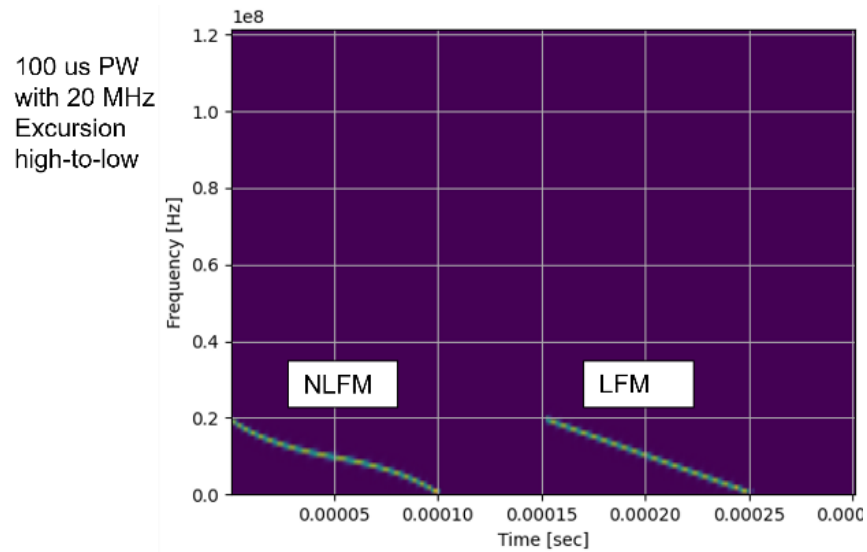


Fig. 12 Vivado simulation of FMGEN-generated high-to-low NLFM and LFM pulses

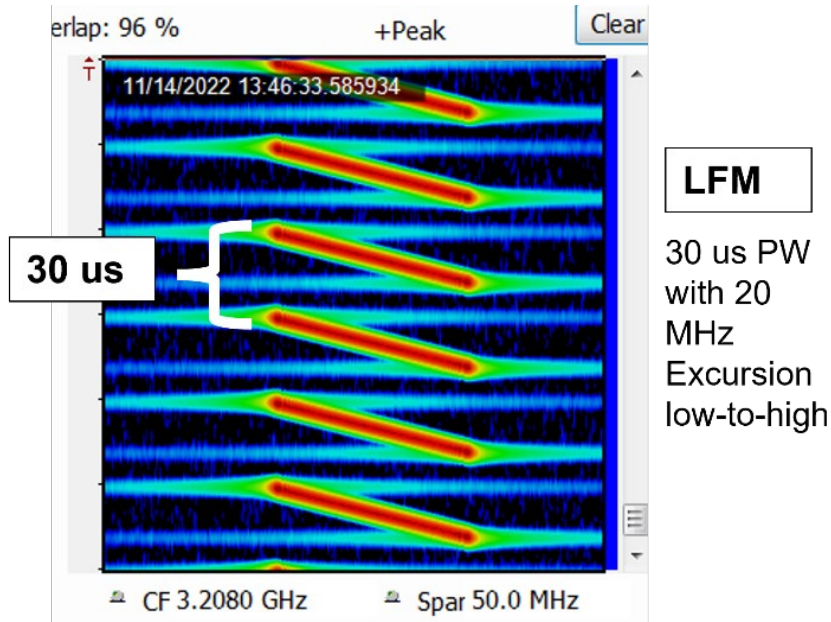


Fig. 13 LFM spectrogram with 30 us PW and 20 MHz FE

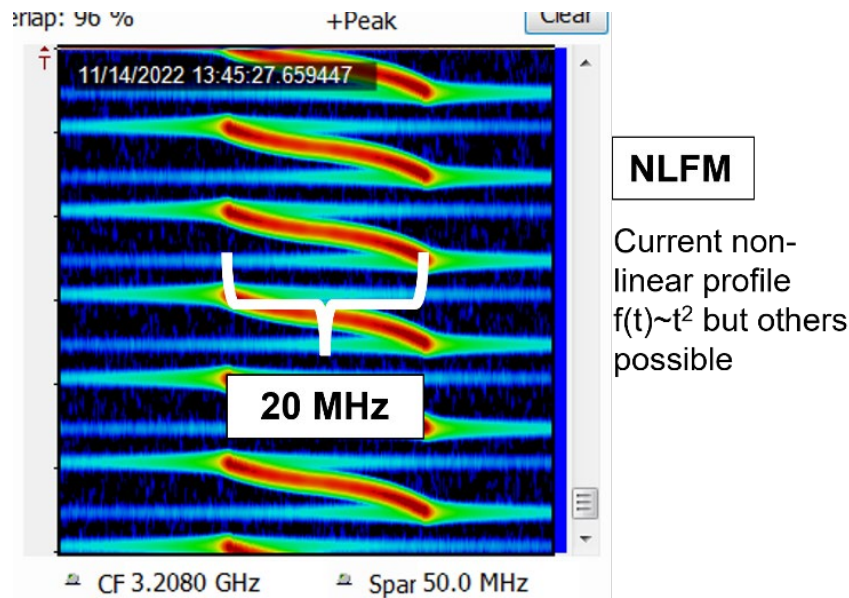


Fig. 14 NLFM spectrogram with 30 us PW and 20 MHz FE

3.2 Binary Phase Shift Keying Generator (BPSKGEN)

The BPSK generator is used to synthesize pulses that undergo 180° phase reversals upon reading binary encodings of 0 or 1. Although arbitrary BPSK parameters can be implemented, this BPSKGEN VHDL IP is designed for Barker coded (BC) pulses. BPSKs with n -bit BCs have n -factor pulse compression for improved range gate and spectral sidelobe reduction approximately $1/n$.¹² BPSKGEN parameters

are user-defined according to Table 6 and include PW, PRI, Barker code length (BCL), and wave frequency increment (WFI). WFI in BPSKGEN is the number of sinusoidal oscillations that occur before a code change.

Table 6 Summary of FMGEN user-defined parameters with example definitions including CLI command sequence

Parameter and command	Abbr/unit	Example	Effect
Pulse width	PW (ns)	s fa0 <cr>	PW = 4000 ns
Pulse repetition interval	PRI (ns)	t 3e80 <cr>	PRI = 16000 ns
Barker code length	BCL	L d <cr>	13-bit BCL
Wave frequency increment	WFI	X 10 <cr>	16 oscillations before code change
Lock BPSK waveform	...	B 0 <cr>	Parameters locked to BPSK waveform 0
Load and run BPSK Waveform	...	b 0 <cr>	BPSK waveform 0 executed on BPSKGEN

Examples of BPSK BCs generated by BPSKGEN are shown in Fig. 15 (time domain) and Fig. 16 (frequency domain). An example of a 13-bit BCL BPSK signal, with code transitions, is shown in Fig. 17. In Fig. 17, WFI = 2.

Of note, BPSKGEN in addition to “lead” and “trail” pulse interrupts provides a third interrupt corresponding to each code change (bpskgen_irq3). These interrupt assertions can be seen as the tick marks at each code change in the bottom of Fig. 17. This interrupt is not used in current ARWG.



Fig. 15 BPSK with 5-bit BC

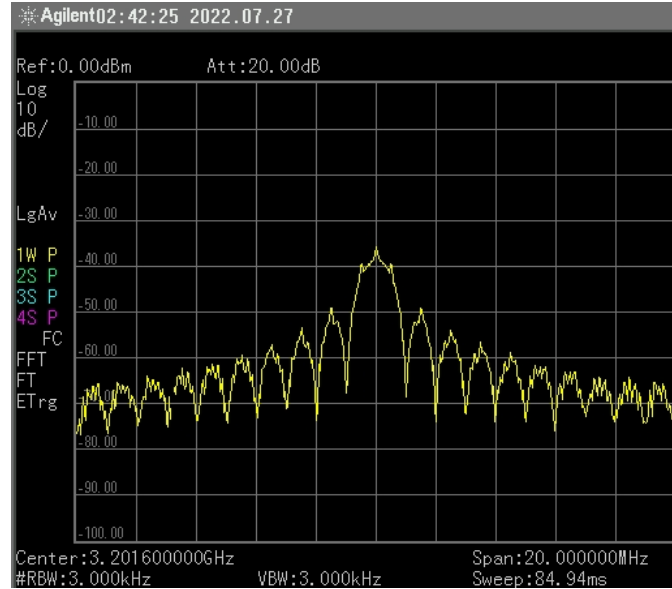


Fig. 16 Spectrum of BPSK with 13-bit BC

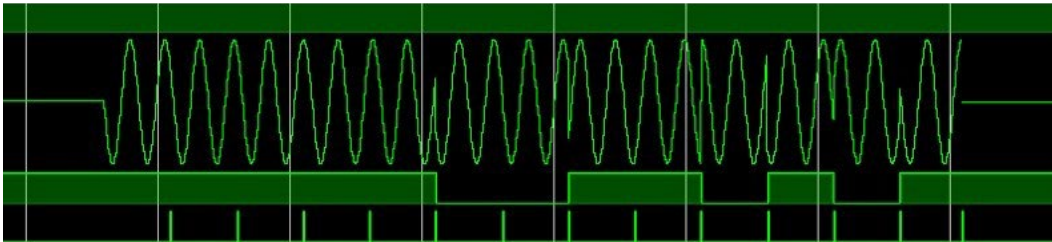


Fig. 17 BPSK with 13-bit BC with code transitions depicted at bottom

3.3 Custom Generator (Custom) or Variable Direct Digital Synthesizer (VARDDS)

The custom generator is used to load user-defined arbitrary waveforms from memory and run them on command. There are two custom generators per channel in this implementation of ARWG—labeled Custom 1 and Custom 2. The custom waveforms, generated and stored by a user software application, are loaded from DDR4 memory to the intended custom generator using the Xilinx SG-DMA IP. Software drivers are adapted from Xilinx embedded drivers.⁴ Only one custom generator can be loaded from memory at a time. The custom selector is used to align AXI Stream pathways in hardware and report the readiness of a custom generator prior to permitting a waveform to run.

The ARWG custom generator is an updated version of the VARDDS used in Diehl and Viveiros.¹¹ The custom selector is likewise borrowed from VARDDSSSEL IP in Diehl and Viveiros.¹¹ Figure 18 shows a configuration of a scheme with a single custom generator. The custom generator uses a ping pong buffer, where the offline

buffer is loaded via the DMA while the online buffer is synthesizing a waveform. A “swap” command is issued by software to the IP to swap the online and offline buffers. Upgrades from VARDDS in Diehl and Viveiros¹¹ to the current custom generator IP include more precise and conforming definitions of PW and PRI, and interrupt signals capable of reporting the start and end of each pulse, that is, “lead” and “trail.” These upgrades were required to facilitate frequency agility and composite pulse group generation, to be detailed in Sections 6 and 7, respectively.

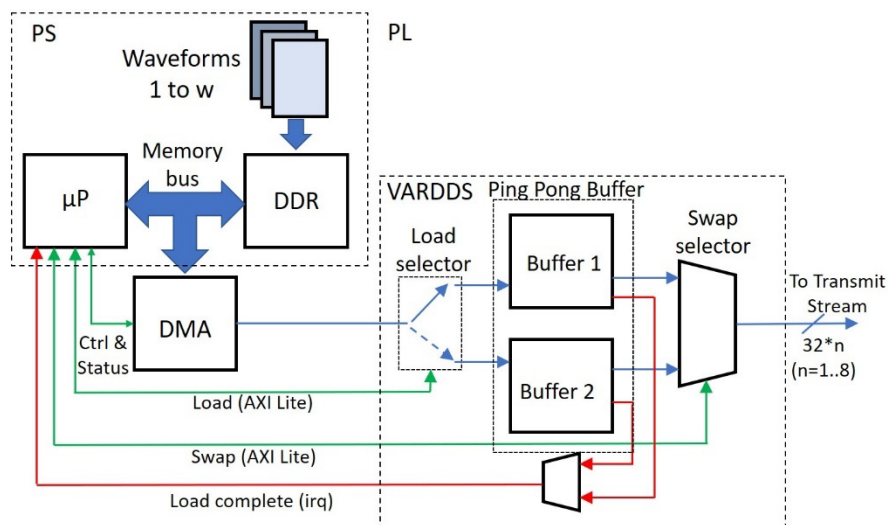


Fig. 18 Connection scheme for VARDDS¹¹

Custom waveforms are stored in memory as an array of 32-bit samples, that is, 16-bit I and 16-bit Q. In current ARWG there are eight pre-defined waveforms included as C header files and stored in an array of structures; this number is arbitrary and can be updated at compile time. Each waveform definition file consists of 16,384 entries; this number is also arbitrary; however, this parameter must be updated at IP synthesis time. In CLI mode, custom waveforms are selected from those residing in memory. In graphical user interface (GUI) mode, however, arbitrary custom waveforms can be generated on-the-fly, for example, from a Python script, and waveforms are dynamically loaded into memory via the Gigabit Ethernet (GigE) interface and run in any custom generator. A sample Python script is included in Appendix B; details about GigE interface and GUI mode will be discussed in Sections 8 and 9.2, respectively.

The custom waveform generator is one of the most powerful capabilities of ARWG. Very complex nonlinear and potentially nonrepeating waveforms can be defined in software and run with low latency, on the order of hundreds of milliseconds from inception to RF available. A sample set of nonlinear waveforms was generated using a Python script; these include polynomial-based NLFM waveforms, for example, $f(t) \sim t^3$ or t^5 , exponential nonlinear waveforms such as $f(t) \sim \exp(1 - t^2)$,

and discontinuous composite nonlinear waveforms, for example, $f(t) \sim t^3 \parallel f \sim \exp(1 - t^2)$. The custom generator can be combined with wideband and long-duration “pulse capture” IP available in other SDEWFoS applications such as RWR and DRFM for a powerful waveform capture and replay capability; waveform capture and replay is not implemented in ARWG and beyond the scope of this report. A set of sample waveforms is shown in Table 7. Spectrograms of ARWG custom-generated signals are shown in Figs. 19–26; spectrograms in Figs. 19–26 correspond to waveforms 1–8, respectively, in Table 7.

Table 7 Custom pulsed waveforms with FM defined according to approximate formula

Waveform	PW (ns)	FE (MHz)	Approximate formula
1	15000	20	$f \sim \exp(1-t^2)$
2	20000	10	$f \sim \exp(1-t^2)$
3	20000	20	$f \sim \exp(1-t^2)$
4	5000	10	$f \sim \exp(1-t^2)$
5	25000	15	$f \sim t^3$
6	5000	5	$f \sim t^3$
7	20000	20	$f \sim t^3 \parallel f \sim \exp(1-t^2)$
8	25000	20	$f \sim \exp(1-t^2) \parallel f \sim t^3$

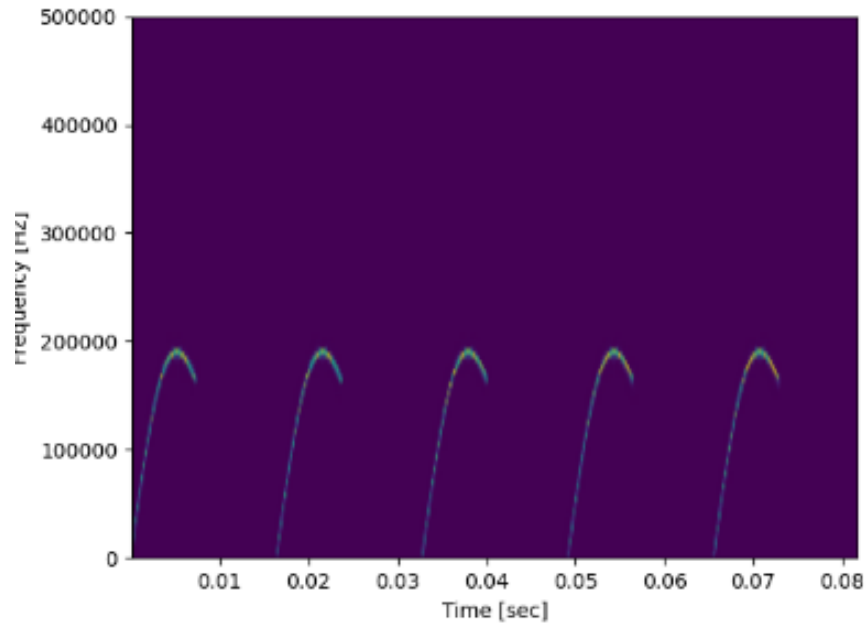


Fig. 19 Custom waveform 1

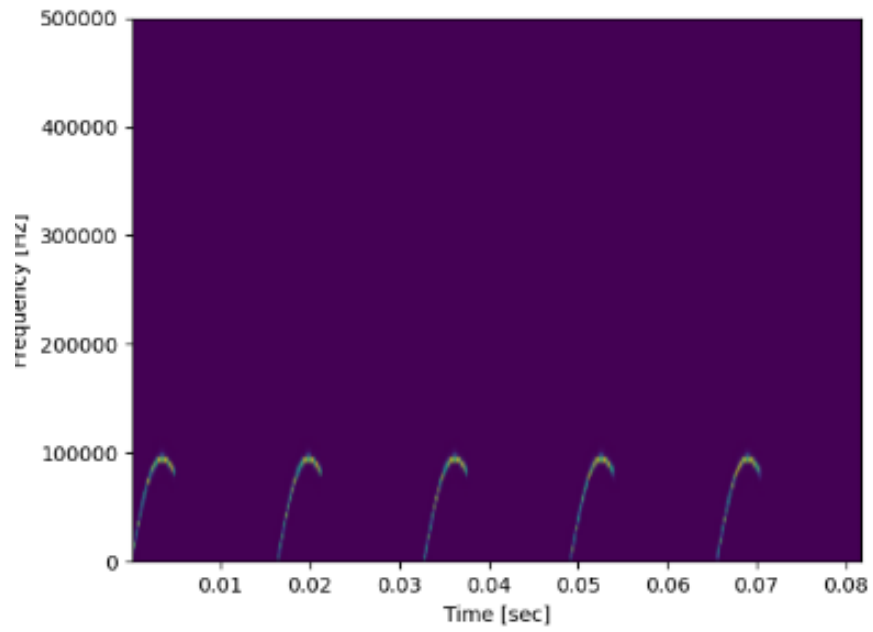


Fig. 20 Custom waveform 2

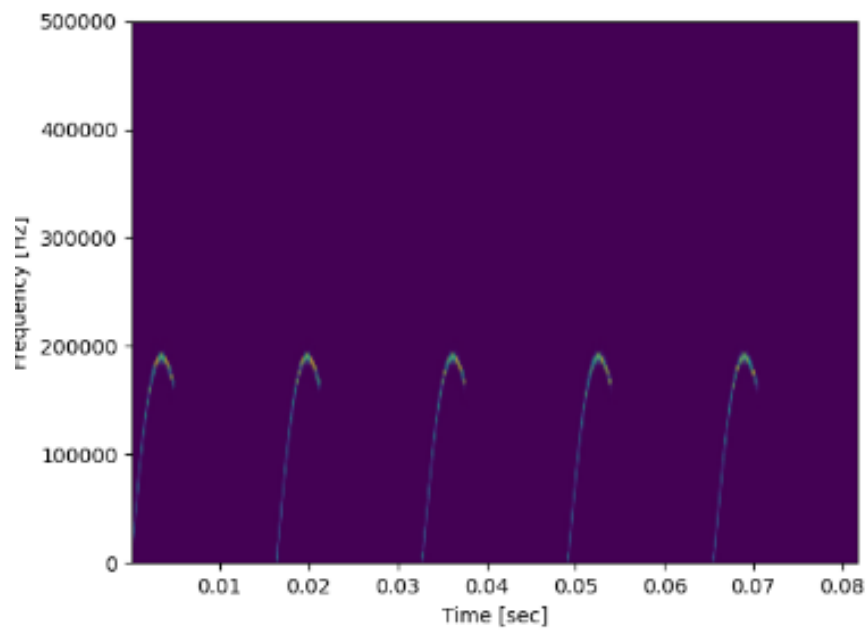


Fig. 21 Custom waveform 3

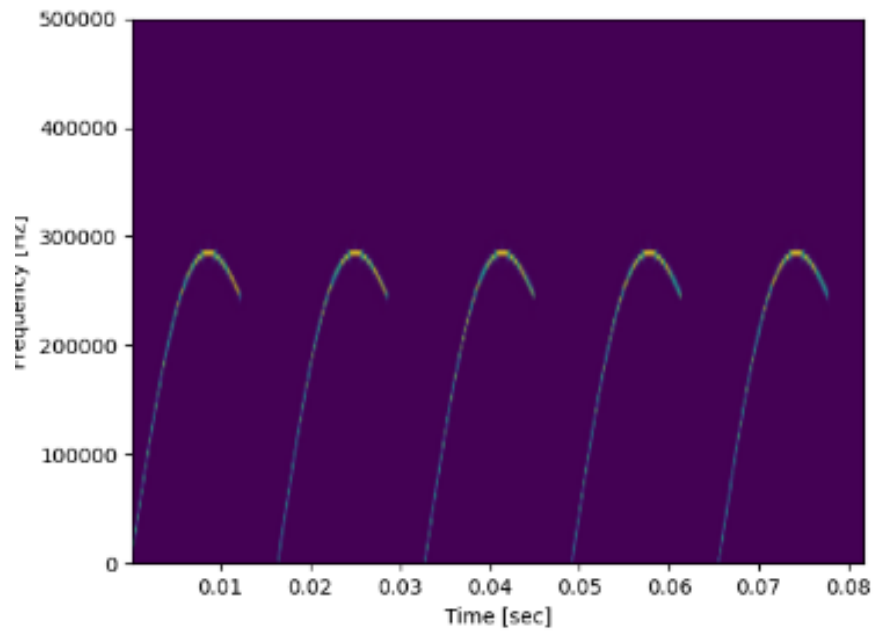


Fig. 22 Custom waveform 4

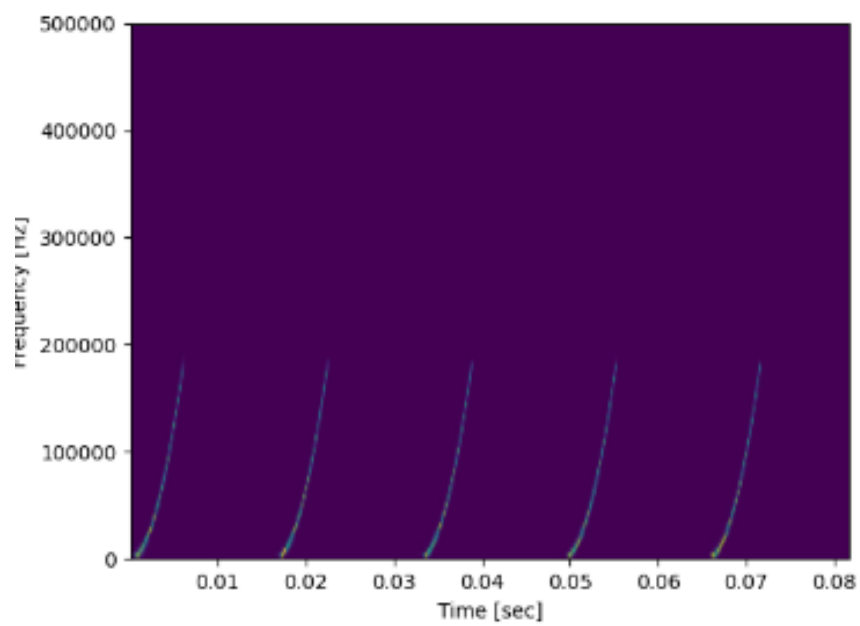


Fig. 23 Custom waveform 5

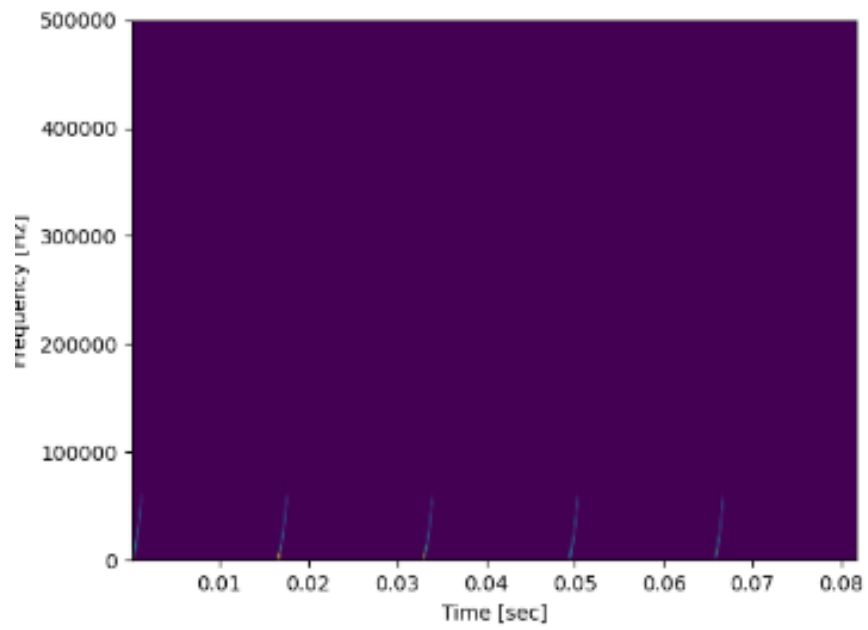


Fig. 24 Custom waveform 6

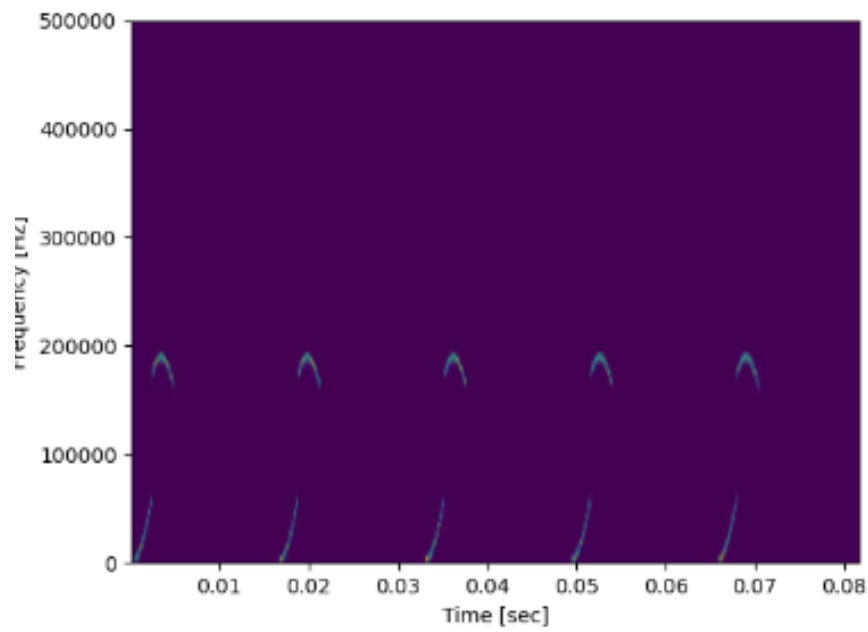


Fig. 25 Custom waveform 7

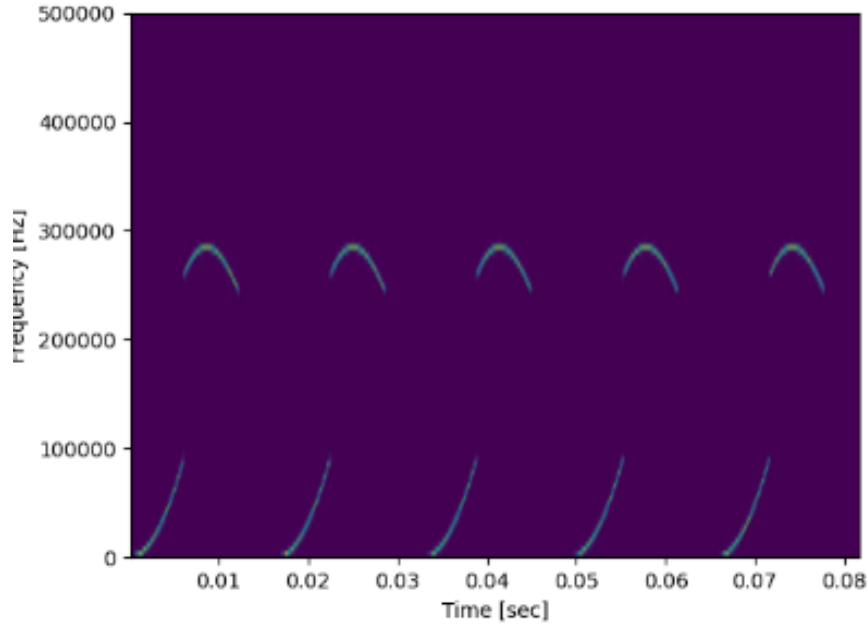


Fig. 26 Custom waveform 8

Custom waveform generation parameters and command sequences are shown in Table 8. In the current ARWG, the tabulated PW must be included as a parameter to program the custom generator, that is, it is fixed in the Python generation script and cannot be mathematically recomputed by hardware. By contrast, PRI is arbitrary and can be user-defined at run time. Additionally, a custom generator instance must be affirmatively selected by running “select custom generator instance” before attempting to load a new custom waveform or deadlock could occur necessitating an application restart.

Table 8 Summary of custom generator user-defined parameters with example definitions including CLI command sequence

Parameter and command	Abbr/unit	Example	Effect
Pulse width	PW (ns)	s fa0 <cr>	PW = 4000 ns
Pulse repetition interval	PRI (ns)	t 3e80 <cr>	PRI = 16000 ns
Select custom generator instance	...	c 0 <cr>	Select custom generator 0
Load new custom waveform	...	A 1 <cr>	Load custom waveform 1 to aligned generator
Load and swap new custom waveform	...	e 0 <cr>	Load custom waveform 1 to aligned generator and swap to active output
Refine custom waveform for PW and PRI	...	J	PW and PRI on standby waveforms are updated and swapped to output on aligned generator

3.4 Signal Selector

An 8-to-1 software-controlled multiplexer in each channel (see Fig. 3) selects the signal generator to be aligned to the digital output path. The upper four selections (4–7) are wired to null, meaning no signal is forwarded. Specifically, the signal is driven to a ground state. Selections 4–7 can be used in future expansions. Table 9 shows signal selector commands.

Table 9 Summary of signal selector commands and effects, including CLI example sequences

Command	Example	Effect
Select generator 0	j 0 <cr>	FMGEN aligned to output
Select generator 1	j 1 <cr>	BPSKGEN aligned to output
Select generator 2	j 2 <cr>	Custom 1 aligned to output
Select generator 3	j 3 <cr>	Custom 2 aligned to output
Select null output	j 4 <cr>	Null aligned to output

4. Signal Modulators

The output of each channel’s signal selector is routed through three serialized modulators as shown in Fig. 3: Phase rotator (Φ), amplitude modulator (G), and noise generator (N).

4.1 Phase Rotator

The phase rotator rotates the phase of the incoming I&Q signal by rotation angle Φ and routes the result to output. The new phase is computed as a complex multiplication on the old phase, where the result is retrieved from an LUT. The LUT approach results in very-low latency (~4 ns phase adjustment) but uses more FPGA LUT resources. The phase input is a unit circle divided into 1024 discrete divisions. The divisions (in hexadecimal) corresponding to quadrants are shown in Fig. 27.

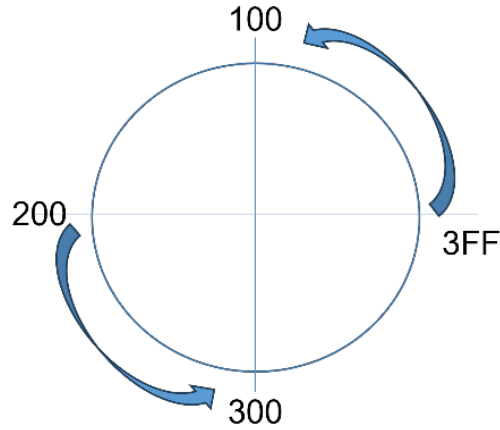


Fig. 27 Phase rotation based on 1024 discrete angles (000 through 3FF hex)

The CLI command sequence for a phase rotation is shown in Table 10. For example, a commanded phase rotation by 0x100 (256) units is equivalent to 256/1024 units, which is a +90° rotation. A software application, or another hardware IP that controls phase, can easily update signal phase at very low latencies, that is, approximately 16 ns from software.

Table 10 Summary of phase rotator commands and effects, including CLI example sequences

Command	Example	Effect
Enter phase offset	a 100 <cr>	Output phase = input phase + 90°

The phase rotator IP was tested by implementing in a test bench that outputs generated signal to baseband and is observed on an oscilloscope with high sampling rate, for example, 20 GSps. Figures 28–30 show an LFM pulse with 4-us PW, 16-us PRI, and 2.0-MHz FE at baseband. The green signal is the phase-shifted signal; the blue signal is the reference. Phase shifts of 0°, 180°, and 90° are shown.

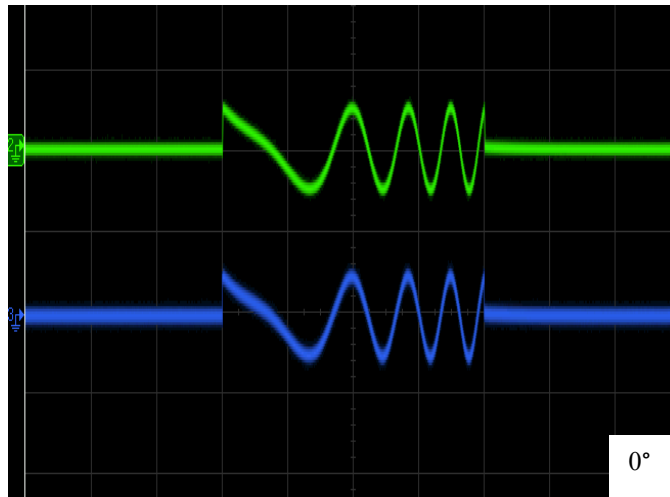


Fig. 28 0° phase rotation

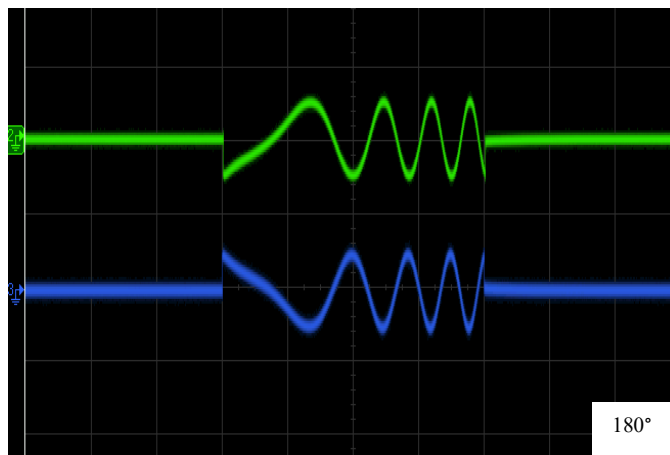


Fig. 29 180° phase rotation

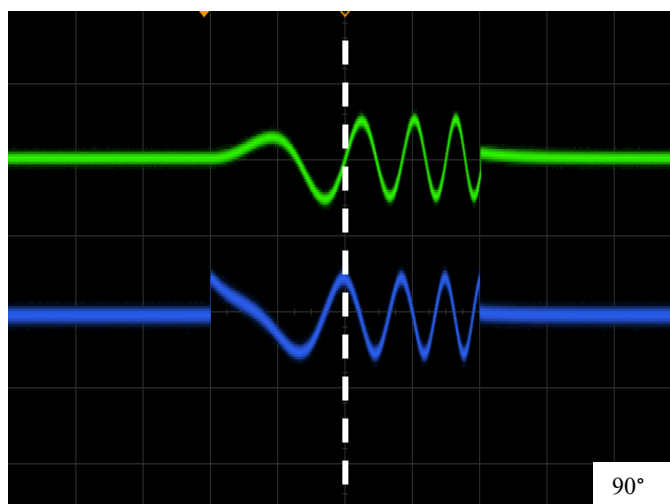


Fig. 30 90° phase rotation

To further verify the phase rotator accuracy and flexibility, a pulse sequence with variable pulse spacing and variable phase rotations was initiated at the baseband. The phase of each individual pulse was manually confirmed on the oscilloscope. The 180° phase shift of the fifth pulse in the sequence in Fig. 31 is shown as an example.

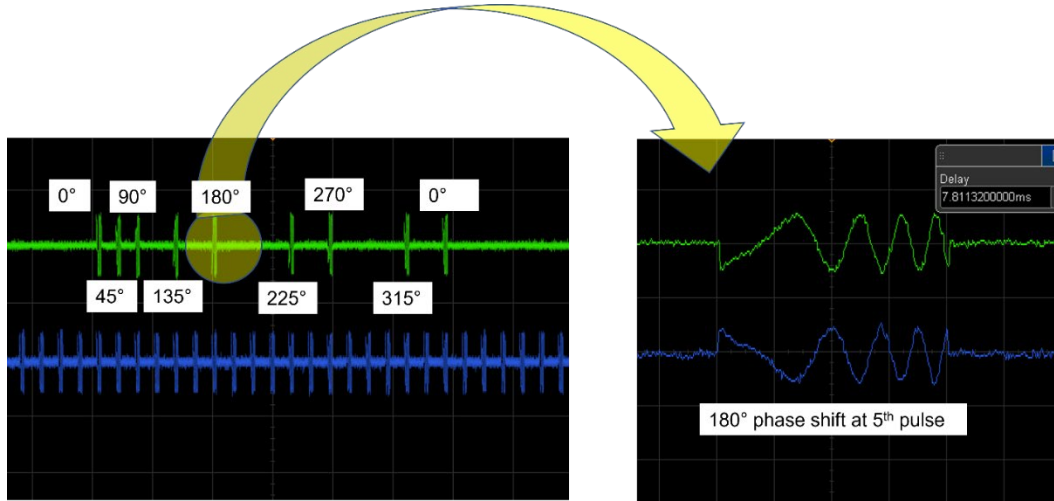


Fig. 31 Pulse sequence used to verify phase rotator accuracy

4.2 Amplitude Modulator

The amplitude modulator (G for “gain”) attenuates any signal in the transmit chain. Attenuation factor G is 0 through F (hex), where 0 is “no attenuation” and F is “maximum attenuation.” The amplitude modulator IP consists of a simple 16-to-1 multiplexer that selects the input signal with 16-G bits of dynamic range. This function can be used to control RF output power, that is, set G to a high attenuation factor until output characteristics are confirmed, then raise back to 0, or can instantiate a waveform where complex dynamic amplitude modulations are required. Attenuation commands and effects are shown in Table 11.

Table 11 Summary of amplitude modulator commands and effects, including CLI example sequences

Command	Example	Effect
Enter attenuation factor	m 1 <cr>	G = 1 (least significant bit of input is truncated)

4.3 Noise Generator

The noise generator adds N bits of pseudo-random noise to the output stream. The mathematical effect is equivalent to additive white Gaussian noise (AWGN). The noise generator consists of an IP based on an FPGA implementation of the Trivium

stream cipher.^{13,14} This Trivium core can output up to 64 pseudorandom bits per clock cycle, meaning that AWGN is produced across the maximum bandwidth of the ARWG channel at current sampling frequency. Noise generator commands and effects are shown in Table 12.

Table 12 Summary of amplitude modulator commands and effects, including CLI example sequences

Command	Example	Effect
Enter noise factor	n 2 <cr>	N = 2 (2 bits of noise are added to 2 LSB of output)

5. Numerically Controlled Oscillator (NCO) Controller

The NCOs used for digital-up-conversion and digital-down-conversion in the RFDC can be controlled through software drivers⁴ or through the real-time signal interface ports for RF-DACs.³ Instantiation of the signal interface ports on the RFDC is selected during RFDC IP customization in pre-synthesis block design. An IP called NCO Controller has been created to implement the NCO real-time signal control, which allows for real-time updates of RFDC CF and phase. One NCO Controller IP is instantiated for each DAC Tile (228, 229, 230, 231). NCO Controller IPs are activated by custom software drivers, although frequency and phase updates could be driven from hardware in the PL, thereby bypassing software entirely. The real-time signal interface ports as implemented in the ARWG are shown in Table 13.

Table 13 NCO real-time interface ports (adapted from Xilinx³)

Port	Width	Description	Implemented
nco_freq	48	Requested NCO frequency setting. This is a 48-bit signed input requesting the NCO frequency, where values range from $-F_s/2$ to $F_s/2$, where F_s is the frequency sampling rate.	Yes
nco_phase	18	Requested NCO phase setting. This is an 18-bit signed number representing requested phase from -180° to $+180^\circ$.	No
nco_phase_rst	1	NCO phase reset used to align phases across the converter.	Yes
nco_update_enable_en	6	Enable NCO register writes.	Yes
nco_update_req	1	Asserted to request update of NCO settings.	Yes
nco_update_busy	1	Asserted while NCO update is in progress.	Yes

The NCO Controller IP is a simple AXI memory-mapped peripheral consisting of input, output, and a state machine. On command, the IP asserts the requested frequency on the 48-bit nco_freq port, and asserts nco_enable_en and nco_update_req. The NCO Controller <x> Start interrupt is asserted, where x

corresponds to the NCO Controller instance 0 through 3. The state machine waits until `nco_update_busy` is deasserted and signals this by asserting the NCO Controller <x> Finish interrupt. The `nco_phase` port is connected to the RFDC, but not currently implemented in the driver (this will be added in a future expansion).

The requested NCO frequency setting is a 48-bit signed integer (C type long long) where values range from $-F_s/2$ to $F_s/2$, and where F_s is the frequency sampling rate. This value is computed in the software driver and forwarded to the IP via an AXI memory-mapped write. The actual NCO frequency change in the RFDC takes some amount of time, which is a function of the internal Xilinx RFDC state machine and the inherent properties of the internal PLL and VCO; this time cannot be accelerated by the RTL designer and represents the lower bound on frequency change latency. During the engineering phase, AXI Timer 0 was used to measure the intervals between NCO Controller Start and Finish interrupts. Our measurements showed that this interval was as low as 900 ns, and never greater than 1,700 ns. Therefore, we made a conservative assumption that an NCO frequency change requires 2 μ s. Frequency change intervals on the Gen1 and Gen3 RFDCs were roughly analogous. Table 14 shows the CLI command sequence necessary to input a frequency to a designated NCO Controller.

Table 14 Summary of NCO Controller commands and effects, including CLI example sequences

Command	Example	Effect
Update NCO carrier frequency	<code>f 0 <cr> 30d400 <cr></code>	Select NCO controller 0 and update frequency to 3,200,000 kHz = 3.200 GHz

6. Frequency Agility

A limited frequency agility mode is implemented in ARWG. The current agility mode is pulse-to-pulse, where the frequency of the selected RFDC oscillator is updated once per PRI. The maximum frequency hop rate is governed by minimum NCO update latency, which is 2 μ s, or 500,000 hops per second. While there are many ways to implement frequency agility, the current ARWG uses feedback from each individual signal generator (FMGEN, BPSKGEN, or Custom). Specifically, the user enters parameters at run time such as the number of channels (currently up to 256) and channel spacing (current maximum is 100 MHz per channel). NCO Controller frequency update parameters are programmed in advance and stored in an array of structures to minimize processor latency when frequency agility mode is selected. When frequency agility mode is toggled to “on,” the assertion of a signal generator pulse trailing interrupt (e.g., FMGEN Trail) will initiate an NCO

frequency update. Therefore, the formula $(PRI - PW) \geq 2 \mu s$ must be satisfied by user input so that the NCO frequency update occurs during pulse off interval. The CLI command sequence for activating frequency agility is shown in Table 15. Spectrogram examples of LFM pulses generated in frequency agility mode are shown in Figs. 32 and 33.

Table 15 Summary of frequency agility commands and effects, including CLI example sequences

Command	Example	Effect
Update number of frequency agility channels	G 4 <cr>	Four frequency agility channels established
Update channel spacing	h 61a8 <cr>	Channel spacing set to 25,000 kHz (25 MHz)
Toggle pulse to pulse frequency agility	F	Frequency agility starts or stops

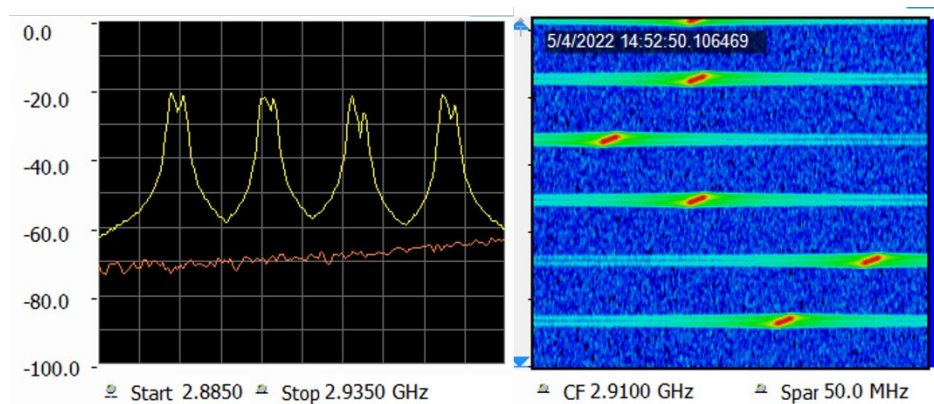


Fig. 32 LFM pulses using frequency agility mode with four channels

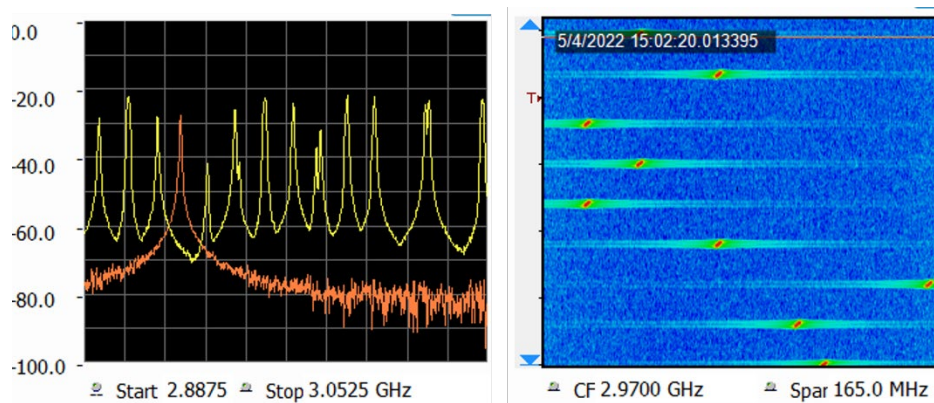


Fig. 33 LFM pulses using frequency agility mode with 16 channels

7. Composite Pulse Sequence Generator

The most powerful of ARWG features is the composite pulse sequence generator. This is designed to allow the user to define multiple waveforms such as FM and

BPSK, and to weave them into heterogeneous groups of pulses that can be started and stopped on demand. Currently, FMGEN and BPSKGEN are configured to use composite pulse sequences. Custom generators will be added in future expansions. A sample pulse sequence with associated terminology is shown in Fig. 34, and the procedure for defining and running the generator is shown in the following algorithm:

- 1 Create waveform (FM or BPSK)
- 2 Create pulse group
 - a Enter waveform type
 - b Enter waveform number
 - c Enter number of pulses in group
 - d Enter off interval after final pulse
- 3 Enter next group into sequence
- 4 Start composite pulse generator
- 5 Stop composite pulse generator
- 6 Append or clear and reenter sequence as desired

The pulse group repetition interval (PGRI) is not explicitly entered by the user but is implicitly calculated as $(\text{no. of pulses in group} - 1) * \text{PRI} + \text{PW} + \text{group off interval}$. The sequence of pulses is rendered in software as an array of structures. At the beginning of each sequence, AXI Timer 1 is set to $(\text{no. of pulses in group} - 1) * \text{PRI} + \text{PW}$, and AXI Timer 2 is set to PGRI. When Timer 1 expires, the respective signal generator is turned off. When Timer 2 expires, the sequence counter is advanced to the next group in the sequence, and the corresponding generator is initiated and started.

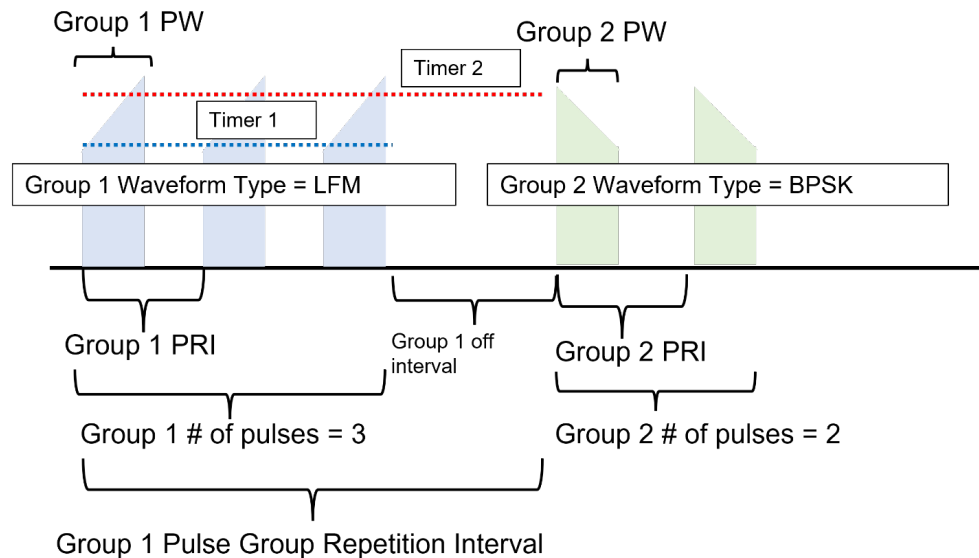


Fig. 34 Scheme for defining and running composite pulse group sequences

The CLI command sequence for aligning and starting the composite pulse group generator is shown in Table 16. An example of a composite pulse sequence generated on ARWG and output to an oscilloscope is shown in Fig. 35.

Table 16 CLI command sequence for aligning and starting the composite pulse group generator

Command	Example	Effect
Enter new group wavetype	Z 0 <cr>	FM wavetype selected
Enter waveform number	w 1 <cr>	Waveform 1 is selected
Enter no. of pulses	x 3 <cr>	Group will have three pulses
Enter pulse group off interval	y 186a0 <cr>	Off interval will be 100,000 ns
Generate and lock new pulse group	z 0 <cr>	Parameters locked as Group 0
Add group to sequence	S 0 <cr>	Group 0 is added to next in sequence
Display group sequence	D	Group sequence displayed, e.g., 2 1 -1 -1 -1
Reset group sequence	R	Sequence reset to -1 -1 -1 -1, etc.
Start pulse group sequence	P	Pulse group sequence started
Stop pulse group sequence	P	Pulse group sequence stopped

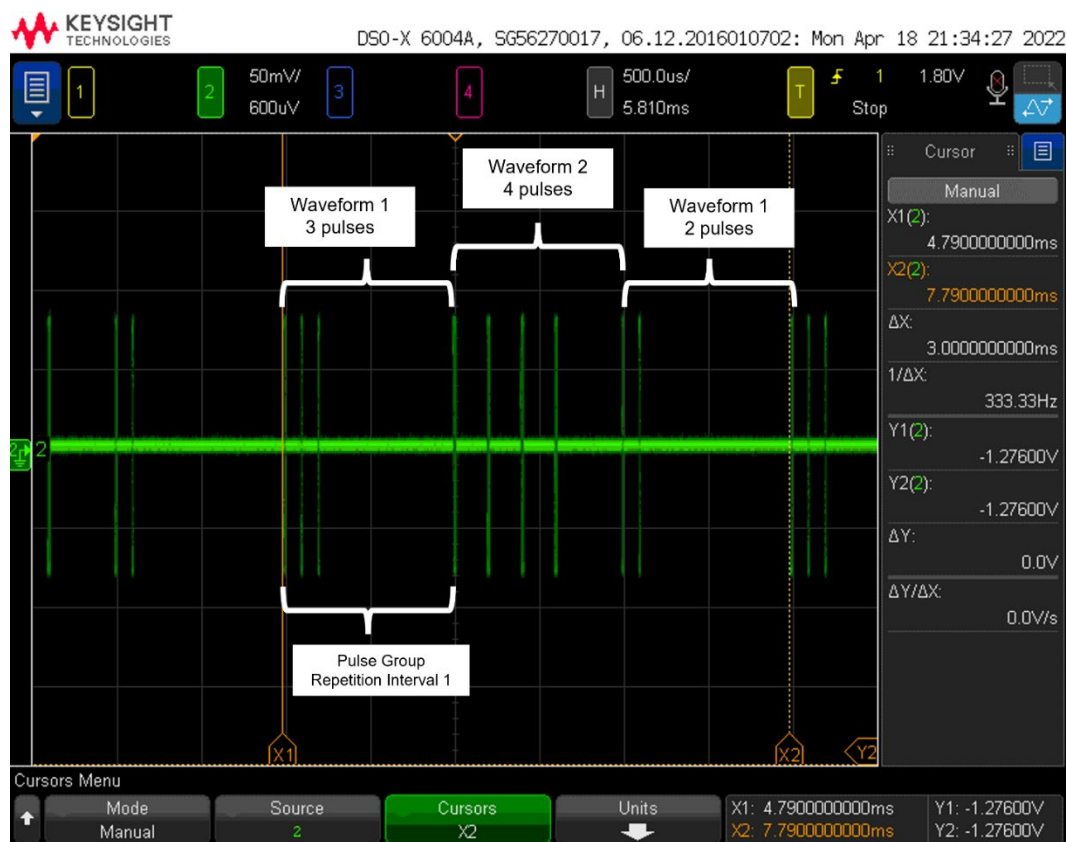


Fig. 35 Sample pulse group sequence created using composite pulse generation

8. Gigabit Ethernet

ARWG, as well as other SFEWFoS personalities, can be connected to a supporting PC using the GigE interface.⁶ The tutorial at Digilent¹⁵ serves as a good template for setting up Ethernet connections on Zynq devices such as Xilinx RFSoc. We use the lightweight internet protocol, transport control protocol (TCP)/IP stack drivers intended for Zynq baremetal implementations.¹⁶ ARWG and PC are then interfaced in a server–client configuration, where RFSoc is the server, and the PC is the client. We use Python applications on the PC, and therefore use the socket library to initiate the client-side connection (Fig. 36).

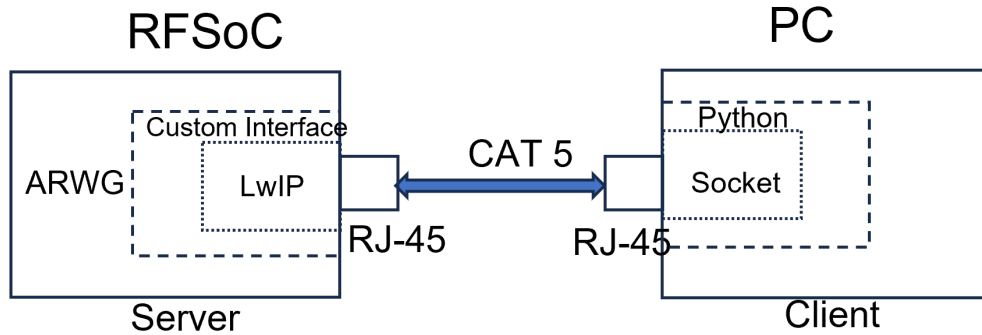


Fig. 36 GigE connecting RFSoc and PC in client–server configuration

Once a TCP/IP connection is established, an application-level protocol is still required. Popular examples include publish–subscribe protocols like Protobuf¹⁷; however, we use a simpler point-to-point protocol for SDEWFoS. Specifically, the custom communications interface in the server activates as a callback function when interrupts are generated on arriving traffic. A 4-byte command field determines how arriving traffic will be processed. The next 4-byte field determines the number of arriving bytes in the packet; subsequent data then arrives in 4-byte fields until complete. Once the expected number of bytes has been received, a status is transmitted back to the server (e.g., “success” or “error”). The applications protocol in the custom communications interface is summarized in Table 17.

Table 17 GigE Applications layer (custom communications interface) protocol

Field	Size (bytes)	Explanation
Command	4	Command from client application
Size	4	Unsigned no. of arriving bytes 1 to 2 ³²
Data	4*n	n 4-byte data words from client
Status	4	Server returns receive status to client

9. User Interfaces

The ARWG has two user interfaces: the CLI and the GUI. There is a near 1:1 correspondence between functionalities of the interface; however, some diagnostic functions are only available in CLI mode, and the ability to generate custom waveforms on-the-fly is only available in GUI mode.

9.1 Command Line Interface (CLI)

In CLI mode, the user types in text commands using the keyboard and responds to requested information in the serial (universal asynchronous receiver-transmitter) application, such as MobaXterm or TeraTerm. The CLI is rigid; commands are case sensitive and backspacing is not permitted. Long numeric entries, such as frequencies or pulse lengths, must be entered in hexadecimal.

The core action mechanism in ARWG (and many embedded designs) is a nonblocking loop that waits for commands coming through the serial interface (or custom interface via GigE). No real-time functions of ARWG are impeded by the nonblocking interface.

The ARWG can be compiled in a “safety” mode. If the ARWG is used to drive actual high-power RF applications radiating to space, the safety mode forces the user to confirm the accuracy of entered commands before they are executed to avoid RF spectrum clearance violations or damage to equipment. The CLI menu is shown in Fig. 37.

Command Menu:
a: Enter waveform phase offset:
A: Load new custom waveform:
b: Load and run new BPSK waveform:
B: Generate and lock new BPSK waveform in table:
c: Select Variable Synthesizer Instance:
C: Update DAC Carrier Frequency:
d: Display group sequence:
D: Display stats:
e: Load and swap new custom waveform:
E: Toggle custom waveform:
f: Update NCO Carrier Frequency:
F: Toggle pulse to pulse frequency agility:
g: Read DAC Carrier Frequency:
G: Update current number of frequency agility channels:
h: Update channel spacing for frequency agility:
H: (Help) Display Command Menu:
j: Select synthesizer output:
J: Refine Custom Waveform (PW and PRI):
l: Load and run FM waveform:
L: Enter BPSK Code Length:
m: Enter new attenuation factor:
n: Enter new noise factor:
p: Start pulse group sequence:
P: Stop pulse group sequence:
r: Reset group sequence:
R: Reset DAC and run diagnostic:
s: Enter new waveform PW in nanoseconds:
S: Add group to sequence:
t: Enter new waveform PRI in nanoseconds:
u: Enter new FM BW in KHz:
U: Enter FM start frequency in KHz:
v: Generate and lock new FM waveform in table:
w: Enter waveform for new pulse group:
W: Enter FM direction (0 = L-to-H; 1 = H-to-L):
x: Enter number of pulses for new pulse group:
X: Enter wave frequency increment:
y: Enter pulse group repetition interval for new pulse group in nanoseconds:
Y: Enter FM linearity (0 = LFM; 1 = NLFM):
z: Generate and lock new pulse group in table:
Z: Enter new group wavetype (0=FM, 1=BPSK, 2=C1, 3 = C2):
Enter command:

Fig. 37 Command line interface menu

9.2 Graphical User Interface (GUI)

The GUI runs in Python on the supporting PC and is implemented by the tkinter library.¹⁸ The primary action in tkinter takes place on buttons, which create events that are processed using Python functions. The basic GUI is shown in Fig. 38. It consists of a number of colored groups of buttons with related functions. The color-coded button groupings are explained in Table 18. Text input boxes are used to input parameters, and accepted parameters are echoed back to the text boxes to

provide ARWG status. Major advantages of the GUI include use of actual full parameters instead of hexadecimal and other encodings in the CLI, all information grouped in logical proximity to operations, and tabulated feedback to the user.

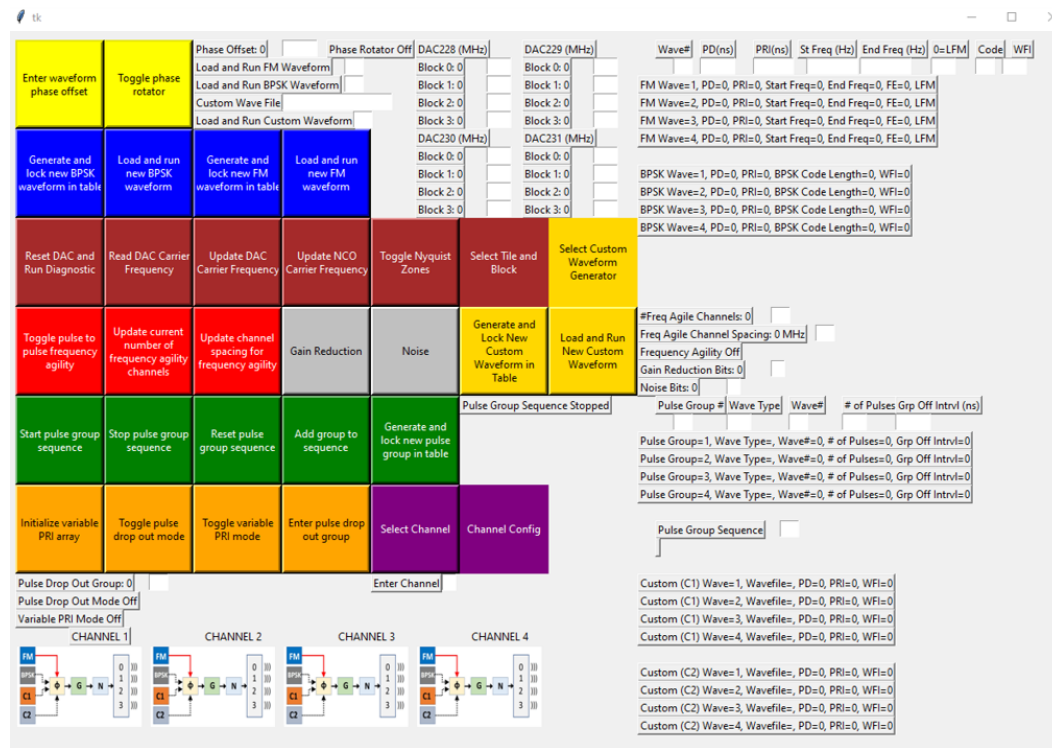


Fig. 38 ARWG full GUI display

Table 18 Color coding of major groupings of GUI buttons

Color	Meaning
Yellow	Phase modulator commands
Blue	Generate, lock, and run FM and BPSK
Brown	Carrier frequency commands & status
Red	Frequency agility
Silver	Modulators (noise and gain)
Gold	Custom waveform generation
Green	Composite pulse group generator
Purple	Channel and generator selection
Orange	Special (outside scope of this document)

Figure 39 is an example of a generate-and-lock FM waveform. The button is shown at left, and the text entry and resulting tabulation of waveform 1 is shown at upper right. The GUI executes on top of the nonblocking command loop and drives the equivalent CLI functions the GigE interface. The lower left shows the serial window (MobaXterm) echoing and executing the FM command. The command is also echoed in the command window of the attached PC; this is shown at lower right.

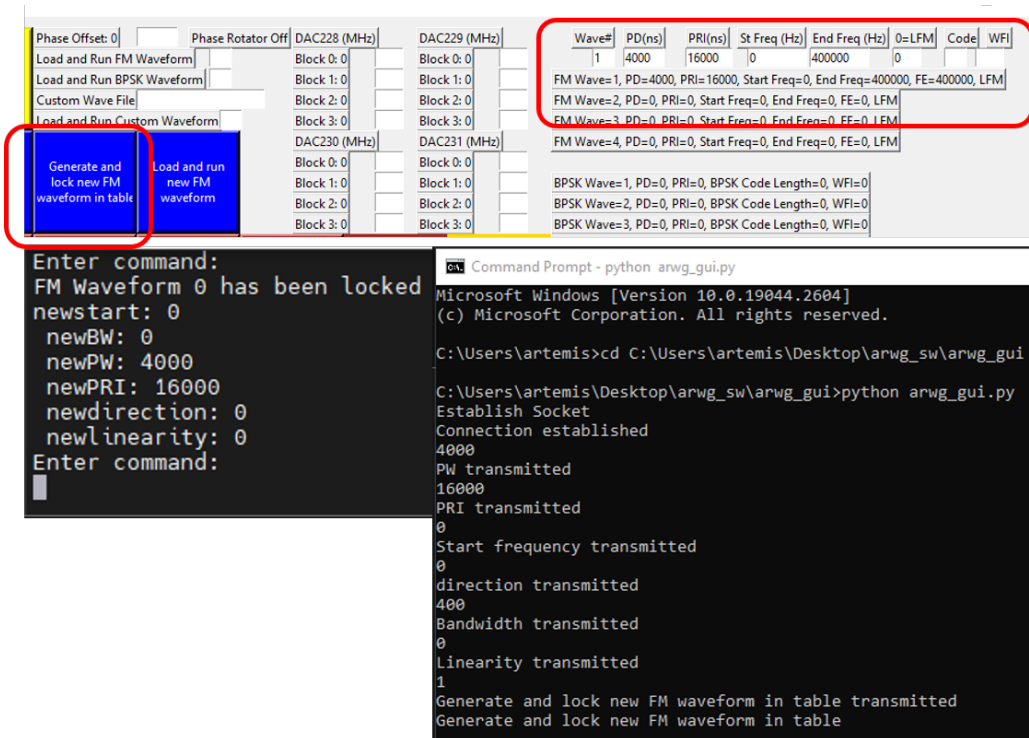


Fig. 39 Generate-and-lock FM waveform from GUI

The GUI is capable of pulling extensive data from the ARWG across the GigE interface into the supporting PC. An example is the Read DAC CF command, which commands RFDC to report all DAC oscillator frequency settings. Settings are retrieved across GigE and displayed in their appropriate text box (Fig. 40).

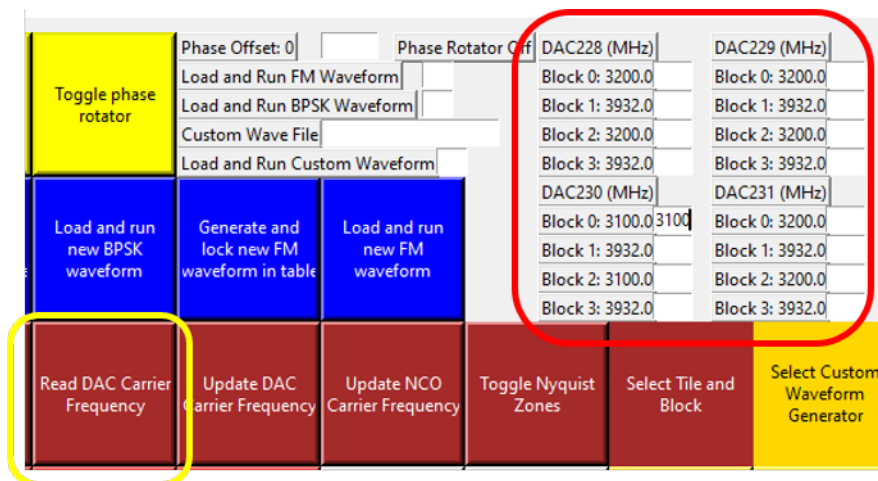


Fig. 40 GUI operation to read all DAC carrier frequencies and display in text boxes

Figure 41 shows an example of custom waveform generation from the GUI. The user either chooses a pre-stored waveform or runs a custom wave generation script in the supporting PC. An example of a Python script to create a custom-generated wave is shown in Appendix B. The resulting waveform is plotted using the matplotlib library; the spectrogram of the resulting custom waveform is shown at lower left. The waveform file “wavefile9.txt” is entered into the text box at the upper left, and the resulting loaded and locked custom waveform parameters are shown in the text box at lower right.

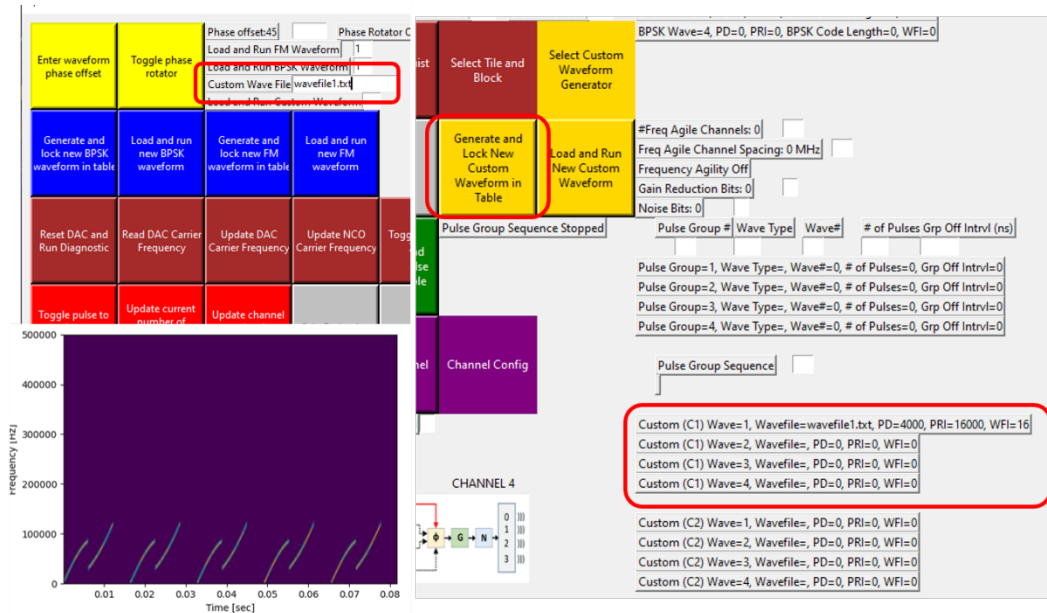


Fig. 41 Generate-and-lock custom waveform using GUI

Figure 42 displays results of instantiating composite pulse groups, a pulse group sequence, and shows the command functions to manipulate pulses.

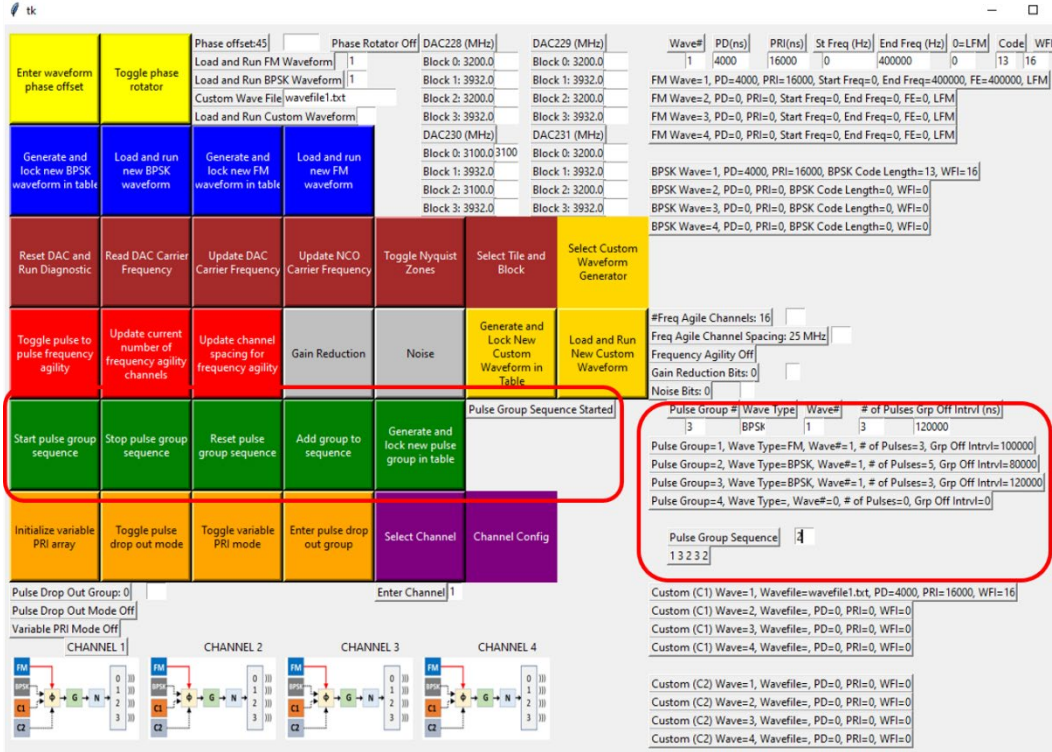


Fig. 42 Composite pulse group generation using the GUI

10. Conclusions and Future Work

We have introduced the Advanced Radar Waveform Generator (ARWG), one of the personalities of the Software Defined Electronic Warfare Family of Systems (SDEWfOS). We discussed the ARWG design based on a baremetal implementation of the AMD Xilinx Gen3 Radio Frequency System on Chip (RFSoc) and discussed its features, such as flexible intrapulse modulations, custom-defined waveforms, frequency agility, multiple modes of signal modulation, and composite pulse group generation. We introduced the two user interfaces: 1) command line interface (CLI), and 2) graphical user interface (GUI) based on the Python tkinter library. Future work will include scaling the ARWG design from one to four channels, fully implementing the custom waveform generators into the composite pulse group generator, and researching novel waveform generation techniques in higher-level software to drive ARWG across the Gigabit Ethernet (GigE) interface.

11. References

1. Xilinx Zynq UltraScale+ RFSoc data sheet: overview (DS889 v1.13). 2022 Jan 7.
2. AMD Zynq UltraScale+ device technical reference manual (UG1085). 2023 Jan 4.
3. Xilinx Zynq UltraScale+ RFSoc RF data converter v2.6 gen 1/2/3/DFE LogiCORE IP product guide (PG269). 2023 May 16.
4. Xilinx embedded drivers [accessed 2023 July 25]. <https://github.com/Xilinx/embeddedsw/tree/master/XilinxProcessorIPLib/drivers>.
5. Vivado AXI reference guide (UG1037) (v4.0). 2017 July 15.
6. Xilinx, ZCU208 evaluation board user guide (UG1410) (v1.0). 2020 July 8.
7. Xilinx, ZCU111 evaluation board user guide (UG1271) (v1.2). 2018 Oct 2.
8. Xilinx, CLK104 RF clock add-on card user guide (UG1437) (v1.1). 2022 Aug 25.
9. Xilinx ZCU208 vitis RFDC driver error [accessed 2023 July 25]. https://support.xilinx.com/s/question/0D52E00006hpSRzSAM/zcu208-vitis-rfdc-driver-error?language=en_US. 2021 Aug.
10. Xilinx, Zynq UltraScale+ RFSoc Gen3: programming the CLK104 module from the RFSoc APU [accessed 2023 Feb 16]. https://support.xilinx.com/s/article/1192842?language=en_US.
11. Diehl W, Viveiros E. Comparison of custom digital synthesizers for advanced radar waveforms in radio frequency systems-on-chip (RFSocS). DEVCOM Army Research Laboratory (US); 2022 June. Report No.: ARL-TR-9471.
12. Wolff C. Intrapulse modulation and pulse compression. radartutorial.eu. [accessed 2023 July 25]. <https://www.radartutorial.eu/08.transmitters/Intrapulse%20Modulation.en.html#p3>.
13. De Canniere C, Preneel B. Trivium specifications. 2005 Apr 29.
14. vtsal/prng_trivium_enhanced [accessed 2023 Aug 2]. https://github.com/vtsal/prng_trivium_enhanced.

15. Digilent//reference. Arty - getting started with microblaze servers [accessed 2023 July 26]. <https://digilent.com/reference/learn/programmable-logic/tutorials/arty-getting-started-with-microblaze-servers/start>.
16. Sarangi A, MacMahon S. LightWeight IP (lwIP) application examples. XAPP1026 (v3.2). Xilinx; 2012 Oct 28.
17. Protocol buffers documentation [accessed 2023 July 26]. <https://protobuf.dev/overview/>.
18. Python Software Foundation. tkinter — Python interface to Tcl/Tk (3.11.4) [accessed 2023 July 26]. <https://docs.python.org/3/library/tkinter.html>.

Appendix A. Vivado IP Integrator Block Design

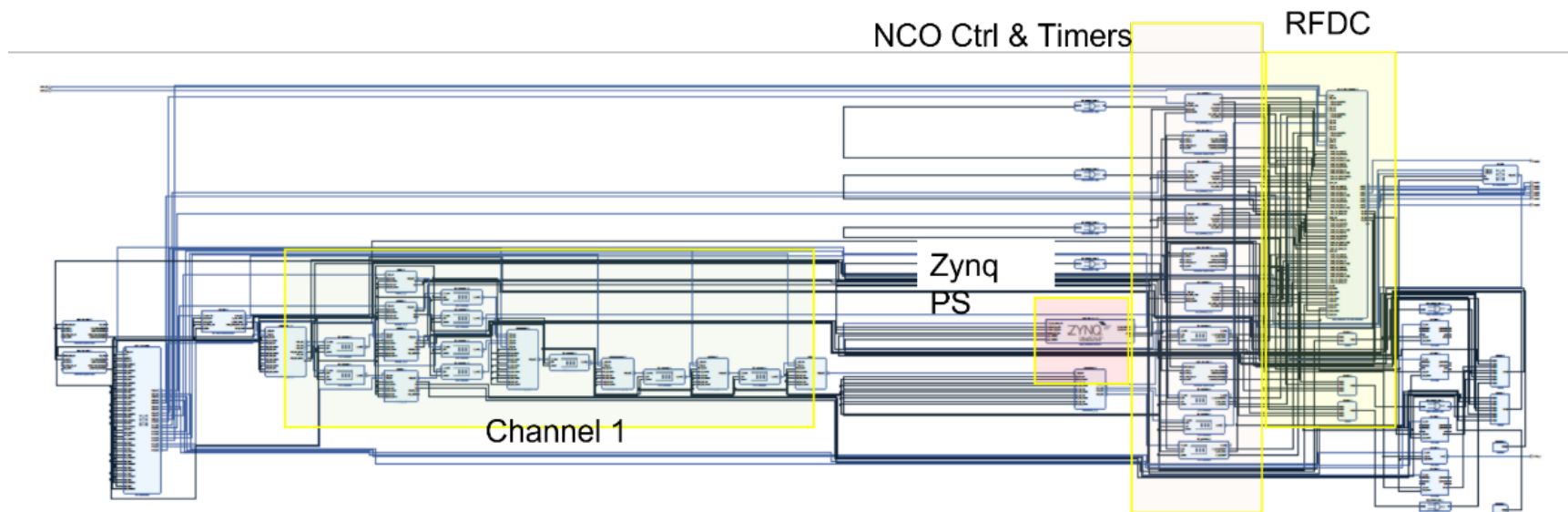


Fig. A-1 Advanced Radar Waveform Generator (ARWG) full block design

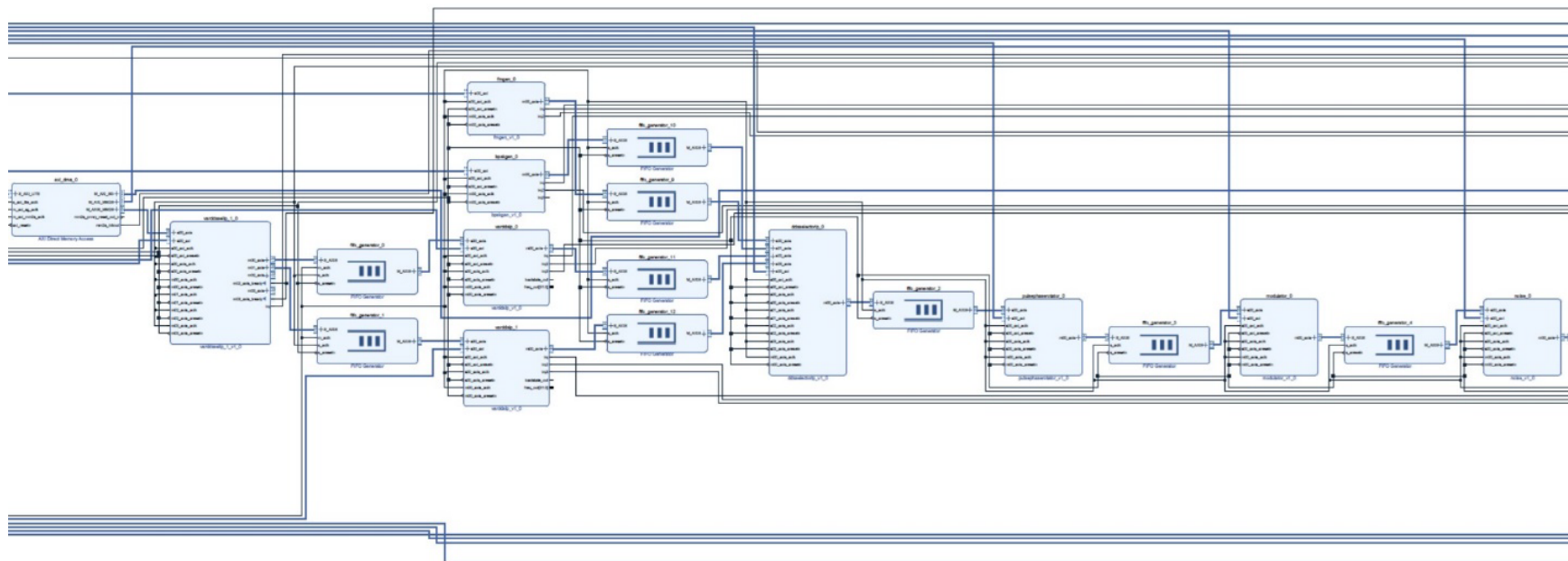


Fig. A-2 Zoom on channel 1 portion of block design

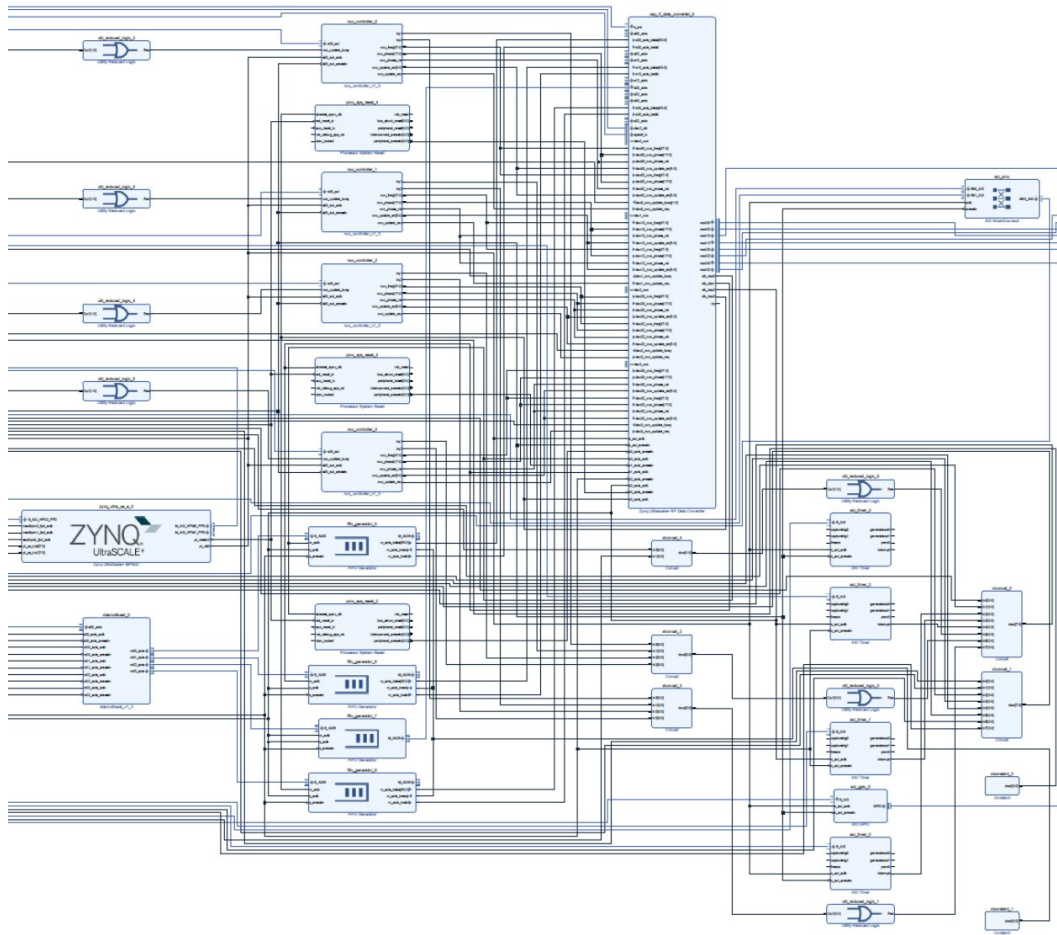


Fig. A-3 Radio Frequency Digital Converter (RFDC), numerically controlled oscillator (NCO), timer, and interrupt concatenation portion of block design

Appendix B. Python Script to Generate Custom Waveform

```

import numpy as np
import matplotlib.pyplot as plt # library for plotting
from scipy.fftpack import fft, fftshift
import random
import math, cmath
from scipy import signal

PMASK14 = 0x0000FFFF
NMASK14 = 0x0000FFFF

# string formatting command

def fp14(x):
    #print(x)
    if (x < 0):
        result = x & NMASK14
    else:
        result = x & PMASK14
    return result

# custom waveform generation parameters

streamfactor = 2 # TDATAWIDTH/2 = 64/32 = 2
N = 16384
f_act = 20E6 # Hz
axifreq = 0.245760 # GHz
pw = 25000 #ns
fexp = 3

period = 1/axifreq
print(period)
pwticks = int(pw/period)*streamfactor
print(pwticks)
A = 32767
tinc = 0.0
periods = 5.0
t = 0.0

coswave = np.zeros(N, dtype=float)
sinwave = np.zeros(N, dtype=float)

yaccum = np.zeros(N*5, dtype=float)

coefffile = open('wavefile9.txt','w')

fmax = f_act * int(pw/period) * (period/1E9) # the maximum frequency
reached at the end of rise
print(fmax)

# 1st half of discontinuous waveform

for x in range(0,int(pwticks/2)):

    tinc = math.exp(float(1-x**2/pwticks**2)) # nlfm exponential based on
    exponential

    coswave[x] = A*math.cos(2*math.pi*fmax*tinc) #2*pi*f*t
    sinwave[x] = A*math.sin(2*math.pi*fmax*tinc) #2*pi*f*t
    cosfx = fp14(int(coswave[x]))
    sinfx = fp14(int(sinwave[x]))
    afxstr = '0x{:04x}'.format(cosfx) + '{:04x}'.format(sinfx) + ',\n' #
    for .h file

```

```

        coefffile.write(afxstr)

# 2nd half of discontinuous waveform
for x in range(int(pwticks/2),pwticks):

    tinc = float((x**fexp)/(pwticks**fexp)) # nlfm exponential based on
    fexp

    coswave[x] = A*math.cos(2*math.pi*fmax*tinc) #2*pi*f*t
    sinwave[x] = A*math.sin(2*math.pi*fmax*tinc) #2*pi*f*t
    cosfx = fp14(int(coswave[x]))
    sinfx = fp14(int(sinwave[x]))
    afxstr = '0x{:04x}'.format(cosfx) + '{:04x}'.format(sinx) + ',\n' #
for .h file

    coefffile.write(afxstr)

cosfx = fp14(0)
sinfx = fp14(0)

afxstr = '0x{:04x}'.format(cosfx) + '{:04x}'.format(sinx) + ',\n' # for
.h file

# open and write results to output file

for x in range(0, N-pwticks):
    coefffile.write(afxstr)

for x in range(0, 5):
    for i in range(0, N):
        yaccum[(x*N)+i] = sinwave[i]

# generate time domain plot of waveform
coefffile.close()
plt.plot(coswave) # plot using pyplot library from matplotlib package
plt.plot(sinwave) # plot using pyplot library from matplotlib package

plt.title('Sine wave') # plot title
plt.xlabel('Time (s)') # x-axis label
plt.ylabel('Amplitude') # y-axis label
plt.grid(b='True',axis='both', which='both')
plt.show() # display the figure

# generate time domain plot of expanded waveform
coefffile.close()
plt.plot(yaccum) # plot using pyplot library from matplotlib package
plt.title('Sine wave') # plot title
plt.xlabel('Time (s)') # x-axis label
plt.ylabel('Amplitude') # y-axis label
plt.grid(b='True',axis='both', which='both')
plt.show() # display the figure

X=fftshift(fft(yaccum,N))
fs = 1E6

# plot N-pt shifted fft result (magnitude only) in frequency domain

NFFT = N
fVals=np.arange(start = -NFFT/2,stop = NFFT/2)*fs/NFFT
plt.plot(fVals,np.abs(X),'b')
plt.title('Xilinx ' + str(NFFT) + '-point FFT - with FFTShift')
plt.xlabel('Frequency (Hz)')
plt.ylabel('|FFT Values|')

```

```
plt.xlim(-fs/2, fs/2)
plt.xticks(np.arange(-fs/2, fs/2+1, fs/5))
plt.show(block=True)

fspec, tspec, Sxx = signal.spectrogram(yaccum, fs)
plt.pcolormesh(tspec, fspec, Sxx, shading='gouraud')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.show()
```

List of Symbols, Abbreviations, and Acronyms

Alg 1	Algorithm 1
ARM	Advanced Reduced Instruction Set Computer Machine
ARWG	Advanced Radar Waveform Generator
AWGN	additive white Gaussian noise
AXI	Advanced Extensible Interface
Balun	balanced–unbalanced
BC	Barker code
BCL	Barker code length
BPSK	Binary Phase Shift Key
BPSKGEN	Binary Phase Shift Keyed Generator
BRAM	Block Random Access Memory
CF	carrier frequency
CLB	Combinational Logic Block
CLI	command line interface
DAC	digital-to-analog
Dir	direction
DMA	Direct Memory Access
DRFM	Direct Radio Frequency Memory
DSP	Digital Signal Processor
EW	electronic warfare
FE	frequency excursion
Fend	end frequency
FM	frequency modulation
FMGEN	frequency modulation generator
FPGA	field-programmable gate array
F _s	sampling frequency

Fstart	start frequency
G	gain
Gen	generation
GigE	Gigabit Ethernet
GSps	Giga samples per second
GUI	graphical user interface
H	high
HAL	hardware abstraction layer
HDL	Hardware Description Language
HW	hardware
I/O	input–output
IP	intellectual property
I&Q	in-phase and quadrature
LFM	linear frequency modulation
LFMGEN	linear frequency modulation generator
L	low
LSB	least significant bit
LUT	look-up table
MHz	megahertz
MSPS	mega samples per second
NCO	numerically controlled oscillator
NLFM	nonlinear frequency modulation
PC	personal computer
PGRI	pulse group repetition interval
PL	programmable logic
PLL	phase-locked loop
PRI	pulse repetition interval

PS	processing system
PW	pulse width
RAM	Random Access Memory
RF	radio frequency
RF-DAC	radio frequency digital-to-analog
RFDC	Radio Frequency Digital Converter
RFSoc	Radio Frequency System on Chip
RTL	register transfer level
RWR	Radar Warning Receiver
SCUGIC	generic interrupt controller
SDEWfS	Software Defined Electronic Warfare Family of Systems
SDR	software-defined radio
SG-DMA	Scatter-Gather Direct Memory Access
SMA	SubMiniature Version A
SPI	serial peripheral interface
SW	software
TCP	transport control protocol
VARDDS	variable direct digital synthesizer
VARDDSSEL	VARDDS Selector
VCO	voltage-controlled oscillator
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
WFI	wave frequency increment
WNS	worst negative slack

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

1 DEVCOM ARL
(PDF) FCDD RLB CI
TECH LIB

2 DEVCOM ARL
(PDF) FCDD RLA LD
W DIEHL
N TESNY