Arbitrary Resampling Filter Design

XAPP1373 (v1.0) February 28, 2022

Summary

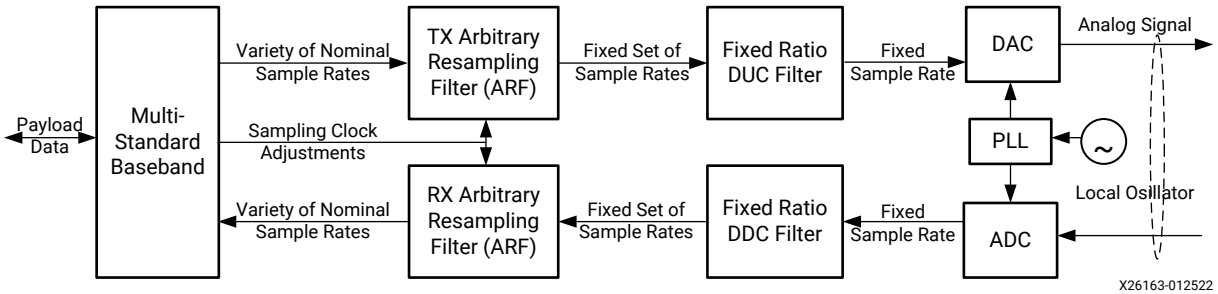
Multi-standard software defined radio systems employ arbitrary resampling filters to support a variety of sample rates. This application note shows the implementation of an arbitrary resampler on a Xilinx® Versal® AI Core device where the controller is in the programmable logic, and the heavy-lifting compute is mapped to the AI Engine. Integration and testing of such a heterogeneous system is simplified by the Xilinx Vitis™ software, which abstracts the processing units as kernels interconnected by AXI buses.

Download the reference design files for this application note from the Xilinx website. For detailed information about the design files, see Reference Design.

Introduction

Modern digital signal processing systems often support multiple communication protocols with various sample rates. However, in the analog front-end, most digital-to-analog (DAC) and analog-to-digital (ADC) converters only work at fixed sample rates. The following figure shows the block diagram of a typical software-defined radio system where a pair of arbitrary resampling filters (ARFs), one in the TX chain and the other in the RX, are employed to support various sample conversion ratios, which can be any real number within a certain range. The output sample rate of an ARF is refined to a small set of fixed data rates that can be efficiently handled by the digital up-conversion (DUC) and down-conversion (DDC) filters. Besides static sample rate conversion, ARFs are widely used for sampling clock error compensation without incurring a high phase noise.

Figure 1: **Multi-Standard Software-Defined Ratio**



The desirable features of the ARF come from its ability to handle dynamic timing offsets of output samples. In other words, the filter coefficients are computed in real-time for given timing offsets rather than in constants. This adds extra complexity to the ARF compared to normal filters. One strategy to lower the computational cost is to place ARFs as close to the baseband as possible, thus reducing the sample rate.

Several options for the ARF implementation in FPGAs have been discussed in *Options for Arbitrary Resamplers in FPGA-Based Modulators* [1]. Xilinx Versal AI Core devices offload the compute intensive part of the filter using AI Engine, while leaving the controller in the programmable logic (PL) for maximum flexibility (see *Xilinx AI Engine and Their Applications* (WP506)). The design of such a heterogeneous system is simplified by the Vitis software, which abstracts the design components as kernels interconnected by AXI buses. This application note provides an example of the design methodology described in *Versal ACAP System and Solution Planning Methodology Guide* (UG1504).

Features

An arbitrary resampling filter is implemented on Xilinx AI Core devices with the following features:

- High performance with 16 taps and a 256x prestored filter coefficient look-up table
- Small footprint with three AI Engines packed in a 3x1 array supporting 250–350 MSPS input sample rates and a fixed output data rate at 500 MSPS synchronous to the output clock
- Sample-by-sample phase adjustment at a refined resolution of 1 ppb
- Deterministic output latency of 1 µs (exactly 500 clock cycles in the output clock domain)
- User-friendly FIFO-like input data and control interfaces
- Fully synchronous output interface with a `solid-High valid` signal and FIFO underflow flags
- Generic Makefile and Tcl scripts reusable by new designs with minor modification

Arbitrary Resampling Filter

Consider the signal waveform shown in Figure 2 where input samples are evenly distributed at an interval T. One desirable output sample is located between x_{n-3} and x_{n-2} with a timing offset *u*, which can be any real number between 0 and 1. For all finite impulse response (FIR) filters, the output can be written as a linear combination of the input samples as:

Equation 1: **FIR Output Linear Combination of the Input Sample**

$$y_n = \sum_{k=0}^{L-1} x_{n-k}\, c_k$$

where L is the number of taps and $\{c_k\}$ is the set of coefficients. In the case of ARF, to account for dynamic timing offsets of output samples, c_k becomes a function of the timing offset u:

$$c_k(u) = f(k + u)$$

where $f(\cdot)$ generalizes the discrete coefficients of a low-pass filter to a continuous function in the domain of real numbers. One method of implementing the continuous function $f(\cdot)$ is to prestore an array $\{F_k\}$ in a memory where

$$F_k = f\left(k/P\right)$$

Then, approximate $c_k(u)$ by a linear interpolation of two nearest prestored values

Equation 2: **Linear Interpolation of Two Nearest Prestored Values**

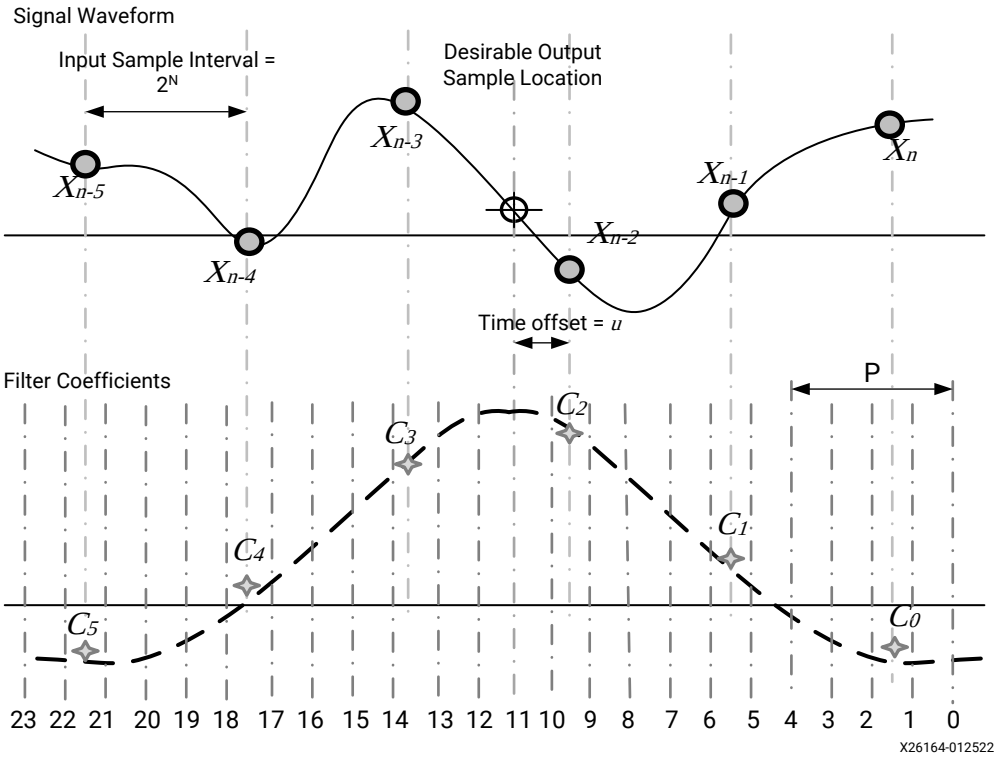
$$\begin{aligned} c_k(u) &\approx (1 - \alpha) \cdot f(k + \lfloor u\,P \rfloor / \mathrm{P}) + \alpha \cdot f(k + (\lfloor u\,P \rfloor + 1)/\mathrm{P}) \\ &= \mathrm{F}_{k\,P + \lfloor u\,P \rfloor} + \alpha \cdot \left(\mathrm{F}_{k\,P + \lfloor u\,P \rfloor + 1} - \mathrm{F}_{k\,P + \lfloor u\,P \rfloor}\right) \\ &= \mathrm{F}_s + \alpha \cdot \mathrm{G}_s \end{aligned}$$

where $[x]$ is the floor function that gives the largest integer less than or equal to x and

$$\begin{aligned} \alpha &= u\,P - \lfloor u\,P \rfloor \\ s &= k\,P + \lfloor u\,P \rfloor \\ G_s &= F_{s+1} - F_s \end{aligned}$$

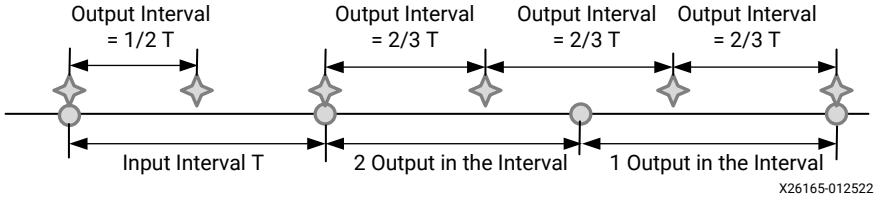
$\{F_k\}$ and $\{G_k\}$ can be implemented by two look-up tables addressed by s. The example in the following figure has $\mathrm{L} = 6$ and $\mathrm{P} = 4$.

Figure 2: **Arbitrary Resampling Filter (L = 6, P = 4)**



For an ARF with a maximum interpolation ratio K, at most ceil(K) new output samples can be computed from one input. The following figure shows the case when K = 2 and either one or two outputs are computed from every new input sample.

Figure 3: **Number of Outputs for Interpolation Ratio up to K = 2**



The complexity of an ARF is dominated by the computation of Equation 1 and Equation 2. The former needs two L real-to-real multiplications for each output sample, and the latter needs an additional L real-to-real multiplications. Because the output sample rate is K times that of input, the total number of real-to-real multiplications is $3 \cdot L \cdot K \cdot \text{Input\_Sample\_Rate}$.

This gives an estimate for the minimum number of AI Engines required by the ARF.

Design Specifications

The ARF should meet the specifications shown in the following table. The input and output sample rates imply the fractional interpolation ratio can be any real number between 1.4286 and 2.0. There can be up to K = 2 samples computed from every new input.

Table 1: **Arbitrary Resampling Filter Design Specification**

Parameter	Value
Range of input sample rate	250–350 MSPS
Output sample rate	Fixed at 500 MSPS and synchronous to the output clock
Interpolation ratio adjustment step	$1/2^{30}$
Number of taps (L)	16
Input/output data format	16-bit I + 16-bit Q
Filter coefficient bitwidth	16-bit
First-in-first-out latency	Fixed at 1 µs (exactly 500 clock cycles in the output clock domain)

The ARF input sample rate is nominal and does not necessarily match the clock frequency of PL logic. Many baseband units process data in bursts, leading to a large fluctuation in the instantaneous input sample rate. Nevertheless, the ARF output should be a continuous data stream, synchronous to the output clock.

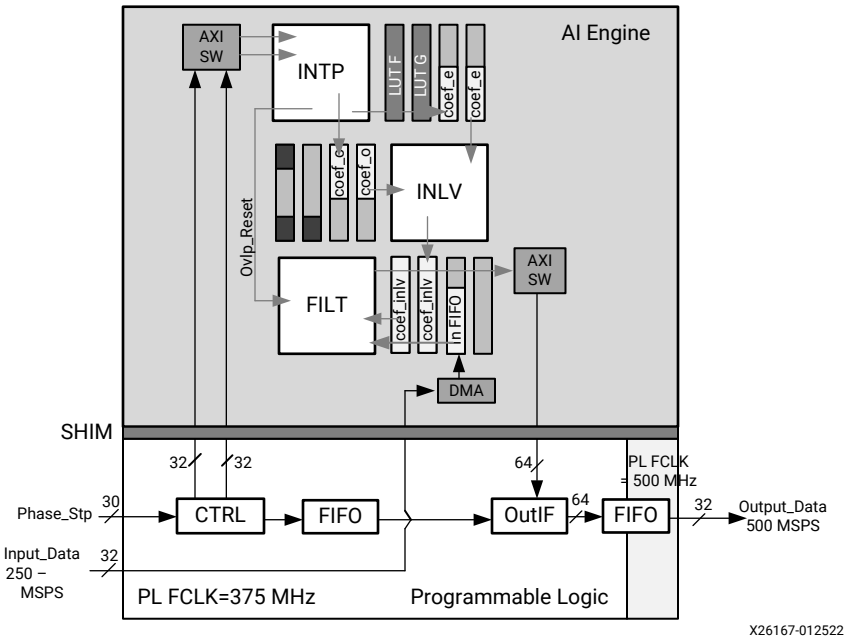
Design Planning

According to the equation $3 \cdot L \cdot K \cdot Input\_Sample\_Rate$, the number of multiplications required to meet the specifications in Table 1 is 3 x 16 taps x 350 MSPS x 2.0 = 33.6G MACs, which exceeds 32G MAC capability of one AI Engine running at 1 GHz. It means that at least two AI Engines are required, for Equation 1 and Equation 2, respectively. Another observation is that the implementation of Equation 2 involves large look-up tables $\{F_k\}$ and $\{G_k\}$, in which all 16 coefficients for one output sample should be read out simultaneously. However, the vectorized implementation of Equation 1 needs the coefficients of four output samples to be interleaved for parallel computation. One more AI Engine should be inserted to interleave the coefficients.

Figure 3 shows the mapping of ARF to the Versal AI Core device. The AI Engines implementing Equation 1 and Equation 2 are labeled FILT and INTP, respectively. The third AI Engine INLV is for coefficient interleaving. The number of output samples computed from every input is fixed to K = 2 in AI Engine, and the OutIF module in the PL removes the invalid data according to a flag generated by the CTRL block, which also computes the phase information {s, α} for coefficient interpolation.

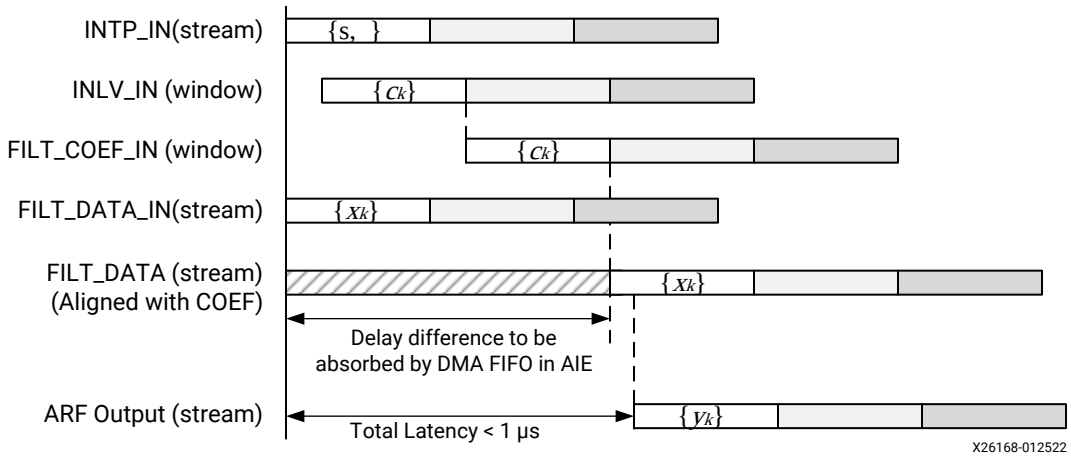
The input and output interfaces of AI Engine strictly follow the AXI protocol where the `Ready` and `Valid` signals might go Low at any time, creating idle cycles. The CTRL module in the PL offers a simple FIFO-like interface for inputs, and the output FIFO removes all the idle cycles to form a continuous data stream in the output clock domain. The AI Engine output sample rate must be K = 2 times that of the input to meet the throughput requirement, so the output AXI bus is 64 bits while the input is 32 bits. For a 350 MSPS input sample rate, a clock of 375 MHz is selected to ensure enough throughput despite the idle cycles. Also, the input phase information might change instantaneously with the data, and {s, α} must be computed on the fly for every input in the PL.

Figure 4: **Partitioning of Arbitrary Fractional SRC Filter onto Versal Device**



The input delays of every AI Engine kernel should be carefully balanced to avoid memory stalls. For example, the following figure shows the coefficients and data inputs to the FILT kernel have a large difference in latency, leading to memory stalls and throughput degradation. To solve this problem, a direct memory access (DMA) FIFO is constructed inside the AI Engine array to absorb the delay differences.

Figure 5: **Balance FILT Input Delays with DMA FIFO**

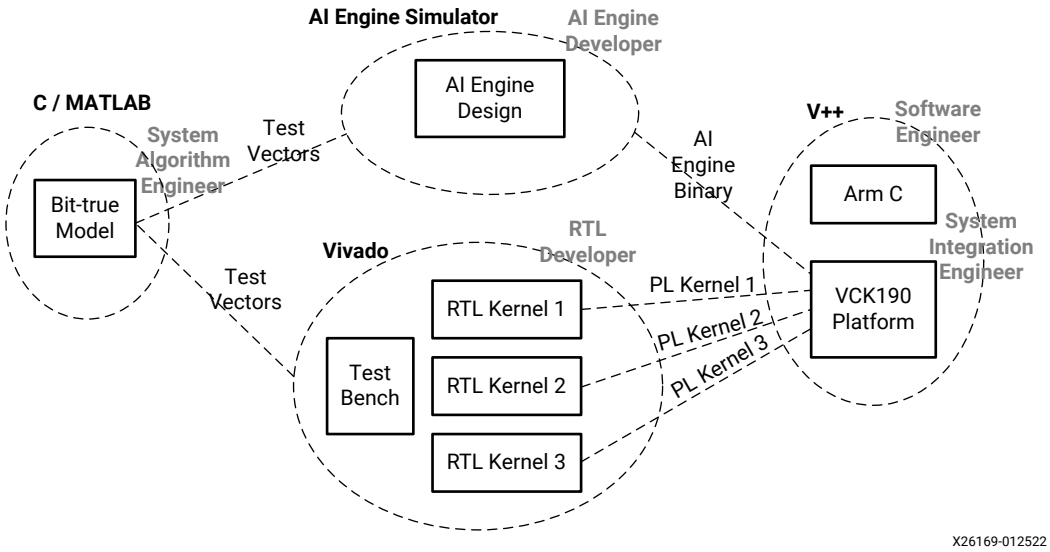


It takes the FILT kernel one clock cycle to compute one output sample, so the peak sample rate can be up to 1 GSPS. Because the target throughput is only 700 MSPS, a margin of 30% can be traded for latency. Figure 5 shows the total AI Engine processing delay is slightly more than twice that of the time to process one window of data. A window of 128 samples translates into 128 x 1/375 = 340 ns latency for the INTP kernel, plus another 340 ns for INLV, and 170 ns for FILT. The total latency is estimated to be 850 ns, leaving 150 ns margin in the 1 µs budget to fill up the output FIFO before the reading starts. The FIFO should be deep enough to accept all prefilled data with some margin to prevent FIFO underflow and overflow when the output is active.

Heterogeneous System Design Methodology

The design methodology described in *Versal ACAP System and Solution Planning Methodology Guide* (UG1504) enables various engineering teams to work on the same design in parallel. The Vitis software abstracts the functional blocks as black boxes, namely kernels, whose interfaces are no more than several AXI buses. Using the bit-true model programmed in MATLAB® or C language, you can precisely determine the behaviors of all the kernels and store the expected data on the AXI buses into text files. This approach effectively decouples the development of AI Engine and RTL.

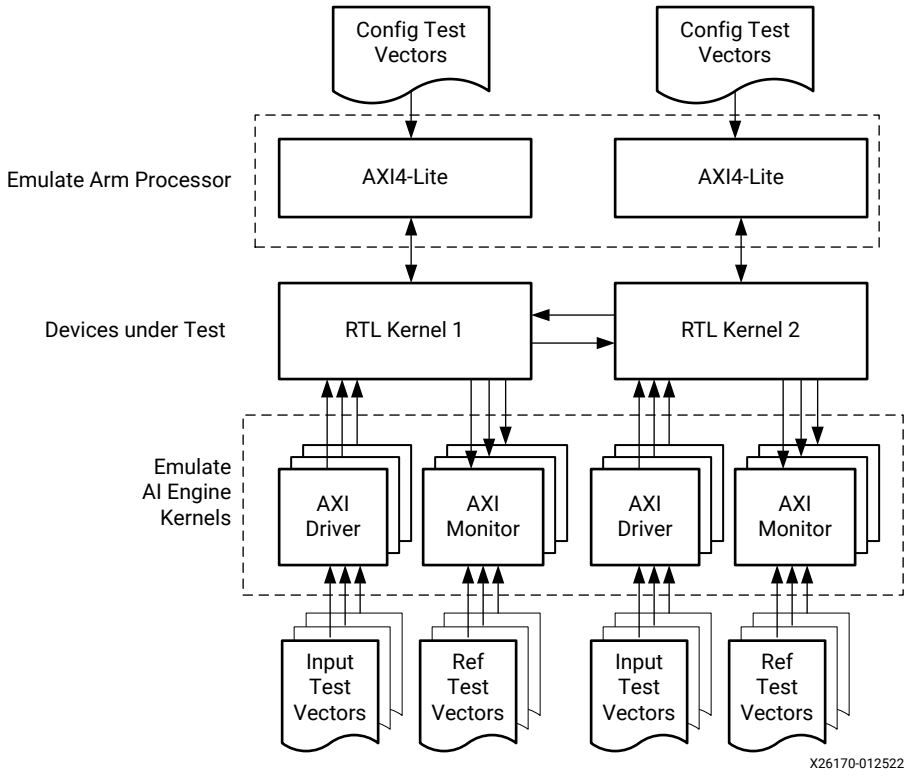
Figure 6: **Heterogeneous Design Development Flow**



From the perspective of AI Engine developers, the whole PL design is reduced to a few input and output AXI streams with the clock frequencies and bus bit widths specified in the AI Engine test bench. Using this information, the AI Engine Simulator drives the input and saves the output accordingly. The AI Engine output data must bit-true match the reference test vectors, and the timestamps saved along with the data give an estimate of the throughput. It is highly recommended to have the AI Engine design fully validated in the AI Engine only simulation environment before integrating into larger systems.

Similarly, from the RTL engineers' point of view, AI Engine kernels are modeled as AXI buses driven or monitored by the RTL test bench. The following figure shows one example of a pure RTL verification environment where two RTL kernels are under test. Many corner cases and extreme conditions are difficult to create with actual AI Engine kernels, and a pure RTL environment is useful to improve the robustness of the RTL design under special circumstances. Also, the simulation of pure RTL is much faster than AI Engine+PL+processing system (PS) co-simulation, leading to shorter turnaround time at the initial stage of development.

Figure 7: **RTL Test Environment**



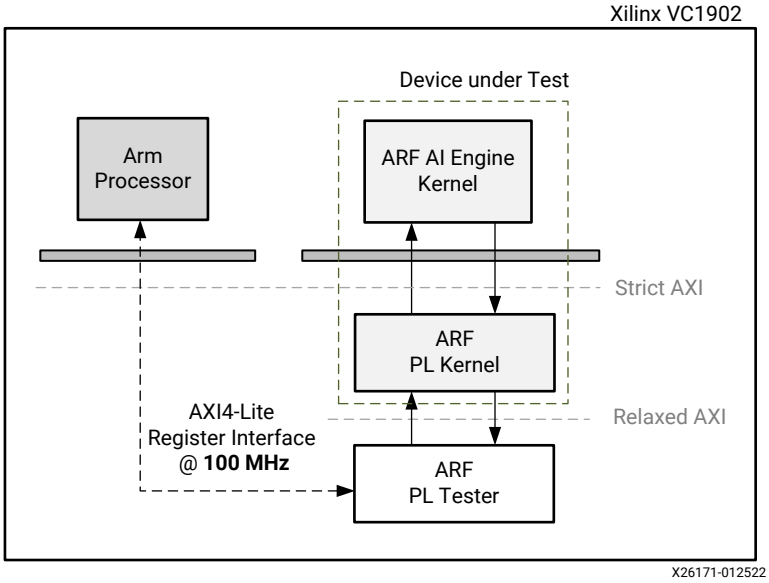
After AI Engine and RTL kernels are independently developed, all the components can be packaged as kernels with only AXI buses for input and output. The Xilinx Vitis compiler is such a productive tool for integration that it only needs the declaration of kernels and a description about the sources and destinations of every AXI bus to automate the connection.

The AI Engine+PL+PS co-simulation should be performed on the integrated design, and the waveform view makes the debug process familiar to traditional RTL engineers. The C program running on the processor controls the test flow, reads back the test results collected by PL kernels, and prints the results via a COM port. During hardware test where the signal waveforms are not available, the printed information becomes the most convenient way to confirm that the design is working correctly.

Design Validation

The heterogeneous ARF design is validated in the Xilinx VC1902 device on a VCK190 evaluation board. The AI Engine and PL portions of the ARF design are packaged as kernels, as is the tester, which drives the input ports of the device under test (DUT) using a prestored stimulus and monitors the output AXI bus with the reference test vector. Throughput and latency are measured by the PL tester and recorded in a set of registers accessible by the processor via the AXI4-Lite interface. At the end of the test, the results are summarized and printed via a COM port.

Figure 8: **ARF Design Validation Environment**



All the kernels can only have AXI interfaces, however, when both source and destination of an AXI bus are PL kernels, users can customize the signal definitions. Besides the AXI buses connected with AI Engine, the ARF PL kernel has the following signals mapped to the AXI interfaces with custom logic.

Table 2: **ARF PL Kernel Signals Mapped to AXI Interfaces**

AXI Bus Direction	Signal Name	Mapping to AXI Signal
Input (375 MHz)	afsrc_in_vld	T_VALID
	afsrc_in_rdy	T_READY
	afsrc_in_soft_reset	T_DATA[63]
	afsrc_in_stp [29:0]	T_DATA[61:32]
	afsrc_in_dat [31:0]	T_DATA[31:0]
Output (500 MHz)	afsrc_out_flags [1:0]	T_USER[1:0]
	afsrc_out_rdy	T_READY
	afsrc_out_vld	T_VALID
	afsrc_out_dat[31:0]	T_DATA[31:0]

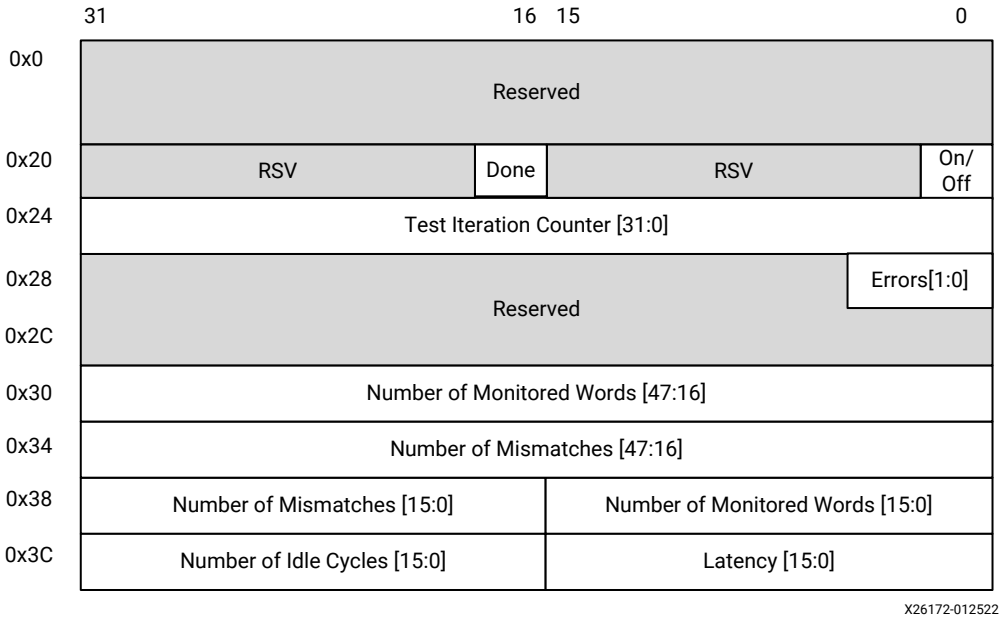
Some details are explained in the following:

- A soft reset is mapped to the most significant bit of the input data bus. It should be asserted before the valid data to do the following:
  - Reset the phase accumulation registers in PL
  - Reset the output FIFOs in PL
  - Clear the overlap memory in AI Engine

- The AXI protocol requires the data transmission to pause immediately after the `Ready` signal goes Low. In the customized AXI interface, the protocol is relaxed to that of a FIFO which honors all write operations until the buffer is full. The backpressure is signaled by the `programmable full` signal asserted when less than 16 samples can be written to the FIFO. This allows the custom logic to flush out the data in a pipeline up to 15 stages.
- The output `Ready` signal serves as a timing reference for the ARF to start output exactly 500 clock cycles after its assertion. This is realized by a carefully controlled output FIFO read signal.
- The `empty` signals of the ARF FIFOs are mapped to T_USER for error detection. When the ARF output is active, a FIFO empty event indicates the output data could be corrupted.

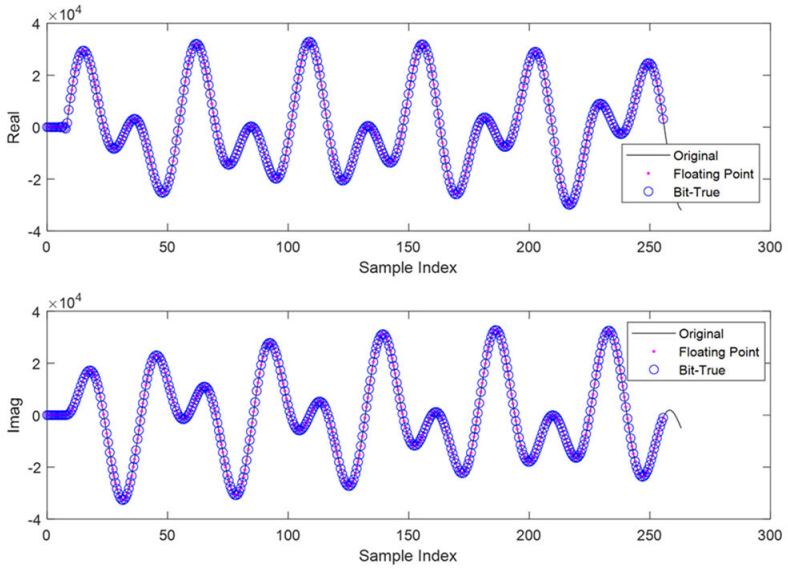
The ARF tester kernel collects the test results to be accessed by the processor via a register map shown in the following figure. There are also fields controlling the test process. Every iteration in the test is 8192 input samples at 350 MSPS, and a maximum of (2^{32} – 1) iterations can last for 8192 x (2^{32} – 1) x 1/350 MHz = 14 hours.

Figure 9: **ARF Tester Kernel Register Map**



A floating-point MATLAB reference model is constructed to ensure the algorithm achieves satisfactory performance. Then a bit-true MATLAB model is developed, and the quantization noise is measured by comparing the output with that of the floating-point model. The following figure is a visual comparison of the input waveform, floating-point resampler output, and bit-true model. They match with each other very well, which suggests a high accuracy. The measured signal-to-quantization-noise ratio (SQNR) is 87 dBc for this test case.

Figure 10: **MATLAB Model Simulation Results**



The test vectors generated by the MATLAB scripts are used for AI Engine simulation and hardware testing. A fractional ratio of 5333/7993 is selected for testing purposes, where 5333 and 7993 are both prime numbers. The input test vector is a repetition of a 5333-sample waveform until the length of AI Engine simulation is reached. The output is expected to be a repetition of 7993 samples, except for the first several samples in the first iteration.

The Makefile includes the commands to run AI Engine simulation and post-process the output data. The test results shown in the following figure suggest that the output of AI Engine kernel bit-true matches the reference test vector, and the target throughput of 700 MHz is achieved with 2% margin.

```
$ make aie
------------------------------------
 Arbitrary Resampler AIE Sim Result
------------------------------------
Throughput = 715.718 Msps
Mismatch   = 0
```

The RTL design is verified in a pure RTL simulation environment with self-checking monitors. Upon the completion of simulation, the test results are output as follows, which suggest the RTL behaviors are as expected.

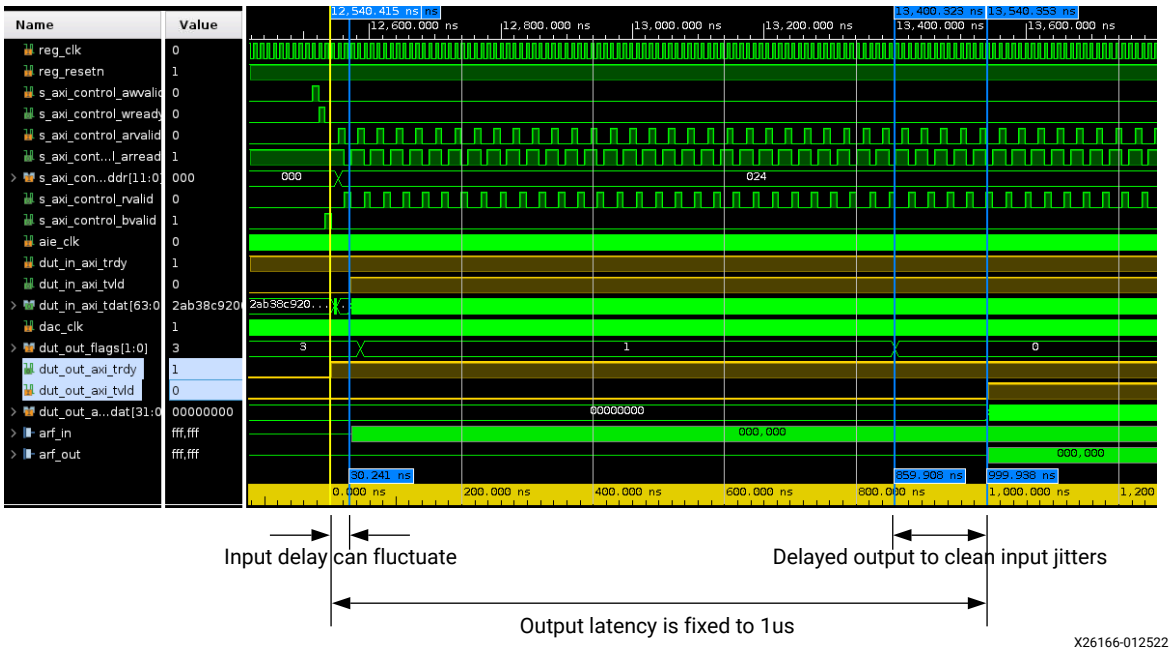
```
$ make rtlsim

SIN Mismatch =     0
AIN Mismatch =     0
DIN Mismatch =     0

Test 0: Mismatch =     0, IdleCycle =    0, Latency = 500 cycles, ErrFlag = 0
Test 1: Mismatch =     0, IdleCycle =    0, Latency = 500 cycles, ErrFlag = 0

*************** TEST PASSED  ***************
```

The AI Engine and PL kernels are now ready for integration. For this design of two PL kernels in three clock domains, the whole system integration is completed with 14 lines of code, as shown in the figure below. A larger design with hundreds of AXI buses can benefit more from this approach because manually connecting thousands of signals in RTL is prone to errors.

```
[connectivity]
# Declare Kernels
nk=tst_arf:1:tst_arf_1
nk=plk_arf:1:plk_arf_1

# TESTER -> PL Kernel
sc=tst_arf_1.arf_in:plk_arf_1.arf_in

# PL Kernel -> AIE
sc=plk_arf_1.aie_sin:ai_engine_0.sin
sc=plk_arf_1.aie_ain:ai_engine_0.ain
sc=plk_arf_1.aie_din:ai_engine_0.din

# AIE -> PL Kernel
sc=ai_engine_0.dout:plk_arf_1.aie_out

# PL Kernel -> TESTER
sc=plk_arf_1.arf_out:tst_arf_1.arf_out

[clock]
# ID=0: 100MHz for Registers
id=0:tst_arf_1.reg_clk

# ID=4: 375MHz for AIE Interface
id=4:plk_arf_1.aie_clk
id=4:tst_arf_1.aie_clk

# ID=3: 500MHz for DAC Interface
id=3:plk_arf_1.dac_clk
id=3:tst_arf_1.dac_clk
```

Debugging with waveform views in a software simulation environment is much easier than doing so directly on hardware with limited visibility. The Vitis compiler supports PS+PL+AI Engine co-simulation and uses the Vivado® simulator as the GUI to display waveforms, on which latencies of various signals can be measured. The ARF output signals in the 500 MHz clock domain (`dut_out_axi_trdy` and `dut_out_axi_tvld` in the figure below) are fine-tuned to have a fixed latency of 1 µs between them. The cross-clock-domain signals and AXI interfaces will have some timing uncertainties. However, they are completely absorbed by the output FIFO and transparent to the custom logic.

Figure 11: **ARF Input and Output Timing Diagram**



After the design passes software verification, more comprehensive and longer tests are performed on the VCK190 evaluation board. By default, VCK190 boards come with VC1902-2MP devices, however, in the test platform, the part number is modified to VC1902-1LLP, which is recommended for customers who prioritize power efficiency. The software running on the Arm® processor starts and stops the test 10 times, from one million iterations (eight billion input samples) in the first test with an increment of 1.2 million iterations (10 billion samples) in each of the following tests. In the end, a short summary is output via the COM port.

```
------------------------------------------------------------------
--        ARBITRARY RESAMPLING FILTER TEST SUMMARY               --
------------------------------------------------------------------
 TestID Latency(us)  Outputs     Idle     Mismatch  Flag Result
------------------------------------------------------------------
   0       1.000      12279095842     0            0   0x00   PASS
   1       1.000      27437128450     0            0   0x00   PASS
   2       1.000      42595161058     0            0   0x00   PASS
   3       1.000      57753242780     0            0   0x00   PASS
   4       1.000      72911275388     0            0   0x00   PASS
   5       1.000      88069307996     0            0   0x00   PASS
   6       1.000     103227340606     0            0   0x00   PASS
   7       1.000     118385373214     0            0   0x00   PASS
   8       1.000     133543442656     0            0   0x00   PASS
   9       1.000     148701475266     0            0   0x00   PASS
------------------------------------------------------------------

PASS!
```

The test result confirms all the design targets have been met:

- All output samples match the reference test vector stored in ROMs.
- A deterministic latency of 1 µs is measured for all the tests.

- No idle cycle is observed in the output data bus, which means the `Valid` signal stays solid High during the test.
- Error flags are not asserted, which means the FIFOs did not underflow.

Conclusion

ARFs have a wide application in multi-rate, multi-standard signal processing systems. The design of ARFs requires a flexible controller in the programmable logic and a heavy-lifting computation engine in the AI Engine. Versal AI Core devices with Vitis software make the design of such complicated heterogenous systems much easier than before. This application note uses a simple ARF design to illustrate the complete tool flow that enables algorithm engineers, AI Engine engineers, RTL designers, and software developers to work together in parallel. This application note provides an example of the design methodology described in *Versal ACAP System and Solution Planning Methodology Guide* (UG1504).

Reference Design

Download the reference design files for this application note from the Xilinx website.

Reference Design Matrix

The following checklist indicates the procedures used for the provided reference design.

Table 3: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Matt Ruan, Hanson He, Allan Zong
Target devices	Versal AI Core
Source code provided?	Yes
Source code format (if provided)	MATLAB script, AI Engine C code, Verilog, and Makefile
Design uses code or IP from existing reference design, application note, 3rd party or Vivado software? If yes, list.	No
Simulation	
Functional simulation performed	Yes
Timing simulation performed?	No
Test bench provided for functional and timing simulation?	No
Test bench format	Verilog and C
Simulator software and version	AI Engine Simulator and XSIM in Vitis 2021.2
SPICE/IBIS simulations	No
Static timing analysis performed?	Yes
Hardware Verification	
Hardware verified?	Yes
Platform used for verification	VCK190

References

This application note uses the following references:

1. C. Dick and F. Harris, "Options for Arbitrary Resamplers in FPGA-based modulators," in ASILOMAR, 2004, pp. 777-781 Vol. 1, available at https://www.semanticscholar.org/paper/Options-for-arbitrary-resamplers-in-FPGA-based-Dick-Harris/2d1b1808b9fd1a66631838f9f3d593f439f4ea91
2. *Xilinx AI Engine and Their Applications* (WP506)
3. *Versal ACAP System and Solution Planning Methodology Guide* (UG1504)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
02/28/2022 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright