Supplementary Materials: FPGA-Based Design of a Ready-to-Use and Configurable Soft IP Core for Frame Blocking Time-Sampled Digital Speech Signals

Nettimi Satya Sai Srinivas, Nagarajan Sugan, Lakshmi Sutha Kumar, Malaya Kumar Nath and Aniruddha Kanhe

S1. Behavioral Simulation of our Proposed Framing Hardware Architecture (FHA)

To verify the functionality of our proposed blocking type FHA (say, the design under test (DUT)), we have performed a simple behavioral simulation using a Verilog testbench. The testbench is configured to use a short finite-length sequence (say, containing 14 unsigned integers, linearly increasing from 1 to 14 and having a word size of $n = 4$ bits) as a test signal for framing with the DUT. The testbench is further configured to transmit the samples of the test sequence to the DUT at a rate, equivalent to a latency of L_w clock cycles, where $L_w = N_f + \delta + 1$ (and $\delta = 4$) that satisfies the practical condition (i.e., $L_w > N_f + \delta$) of our FHA to perform proper framing.

Conversely, the DUT is configured to use a 16-by-4 (depth-by-width) single-port random access memory (SP-RAM) that supports to obtain frames having $N_f \in [2, 16]$ samples (as determined by the `N_f` input). Further, the resulting frames may be non-overlapped ($N_o = 0$) or overlapped by $N_o \in [1, N_f - 1]$ samples (as determined by the `N_o` input). To verify the functionality of the DUT for the mentioned configuration, we have considered a test case, wherein we set the configuration data (N_f and N_o) for the `N_f` and `N_o` inputs to 7 and 2, respectively, and the expected frames from the test signal are provided in Listing SA1, presented inAppendix SA.

Finally, the behavioral simulation is performed, and the functionality of our FHA (DUT) is verified for the test case by comparing the frames obtained from the simulation with the expected frames provided in Listing SA1. We found that the test case is passed, and our FHA (DUT) is functioning as intended. Refer to supplementary materials, specifically Figure S1 (Part-1–3), illustrating the signal timing waveforms captured during the behavioral simulation of our FHA.

S2. Implementation, Testing and Validation of our Proposed Frame Blocking Intellectual Property (IP) Core on a Field-Programmable Gate Array (FPGA)

We present a simple block design (illustrated in Figure S1) to implement, test and validate our proposed frame blocking IP core on an FPGA target, specifically the Avnet® ZedBoard™ [1], having the xc7z020clg484-1 device from the Xilinx® Zynq™ 7000 APSoC (all programmable system-on-chip) family [2]. The block design uses three Xilinx® IP cores, viz., clocking wizard [3], virtual input/output (VIO) [4] and integrated logic analyzer (ILA) [5], and two custom-made user IP cores, viz., data source and pulse generator, in addition to our frame blocking IP core. These IP cores are discussed in detail as follows:

Clocking Wizard IP Core:

It simplifies the creation of customized clock circuits as per the clocking requirements. In our block design, we have configured the MMCM (mixed-mode clock manager) primitive to generate an output clock (`clk_out1`) of frequency $f_c = 50\ MHz$, with reference to an input clock (`clk_in1`) of frequency $f_c = 100\ MHz$. The block design is operated within a single clock domain, wherein the rest of the IP cores are synchronized with the generated clock (`clk_out1`). The input and output (I/O) ports of the clocking wizard IP core are described in Table S1.

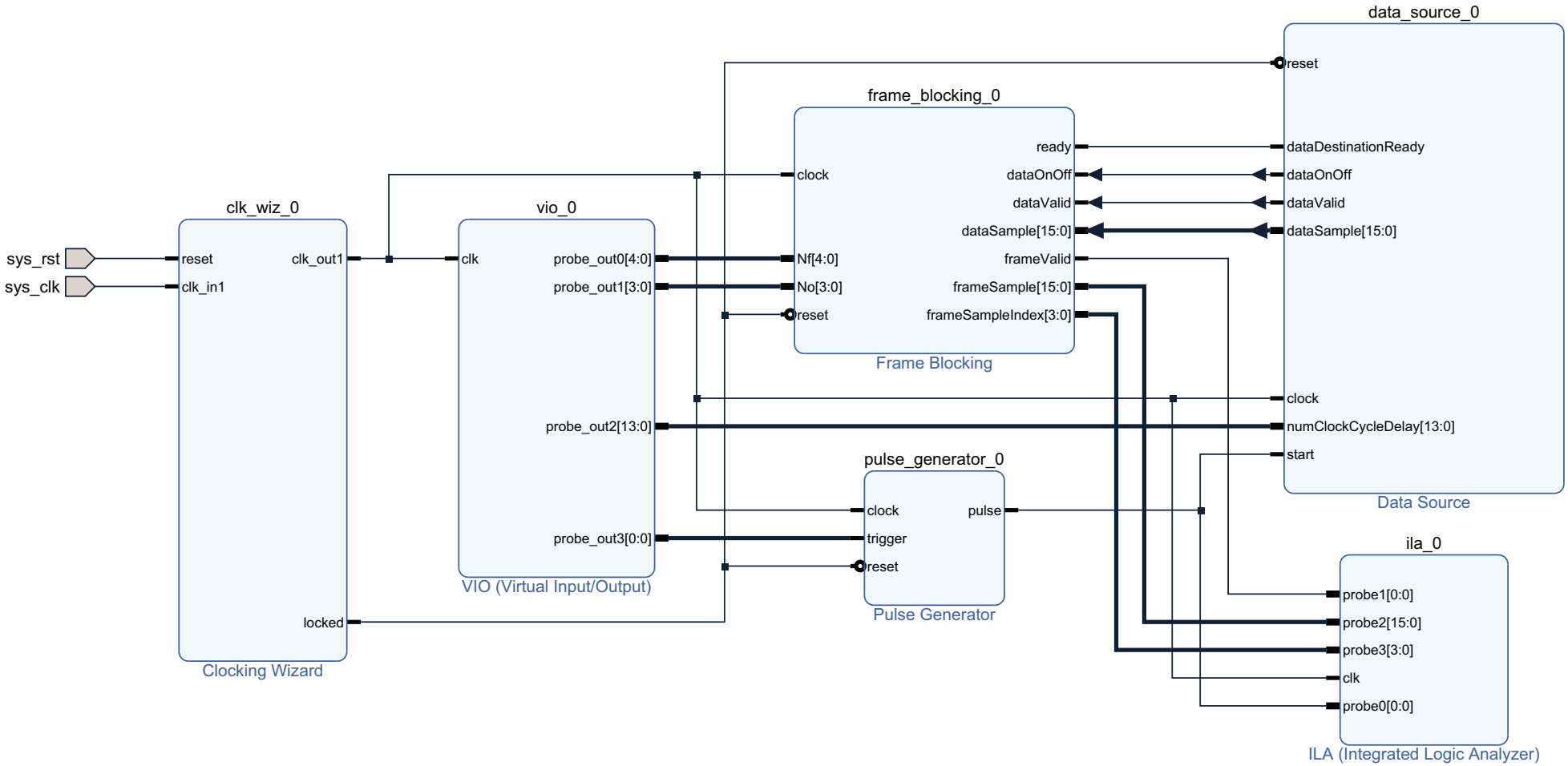

Figure S1. Block design to implement, test and validate our proposed frame blocking IP core on an FPGA target.

Table S1. I/O ports of the clocking wizard IP core.

Port	Direction	Width (Bits)	Description
`clk_in1`	Input	1	Clock port connected to a single-ended primary clock source[1] (`sys_clk`).
`reset`	Input	1	Active-high reset port connected to a reset source[2] (`sys_rst`). When asserted, it asynchronously clears the internal state of the MMCM primitive and causes the primitive to re-initiate the clock-locking sequence when released.
`clk_out1`[3]	Output	1	Output clock of the clocking network.
`locked`[4]	Output	1	When asserted, it indicates that the output clock (`clk_out1`) is stable and usable by the downstream circuitry.

[1] An on-board 100 *MHz* oscillator. [2] E.g., an on-board user push button. [3] To drive the clock input ports of the downstream circuitry. [4] To drive the active-low reset input ports of the downstream circuitry.

Table S2. I/O ports of the data source IP core.

Port	Direction	Width (Bits)	Description
`clock`	Input	1	Clock port connected to a single-ended clock source[1].
`reset`	Input	1	Active-low reset port connected to a reset source[2]. When asserted (to a low logic state), it synchronously resets the IP core and causes the core to switch to an idle mode when released.
`numClockCycleDelay`	Input	14	Data port to load the configuration data, L_w, where $L_w \neq 0$.
`start`	Input	1	When asserted (for one clock cycle), the IP core switches from the idle mode to the transmit mode. The IP core transmits the pre-stored samples in the transmit mode and switches back to the idle mode upon completion.
`dataDestinationReady`	Input	1	Determines the status of the data destination node[3] as idle or busy. The IP core checks this status upon switching to the transmit mode. When asserted, the IP core begins data transmission.
`dataOnOff`[4]	Output	1	Asserted when the IP core begins data transmission. Remains asserted until the IP core completes the data transmission and switches back to the idle mode.
`dataSample`[5]	Output	16	Serial data port to transmit pre-stored samples.
`dataValid`[6]	Output	1	Asserted (for one clock cycle) at every instant when the IP core transmits a sample via the `dataSample` output.

[1] `clk_out1` output port of the clocking wizard IP core. [2] `locked` output port of the clocking wizard IP core. [3] The frame blocking IP core. [4-6] To drive the `dataOnOff`, `dataSample` and `dataValid` input ports of the frame blocking IP core, respectively. Note: The configuration data for the `numClockCycleDelay` input must be loaded before the `start` input is asserted. Unless otherwise specified, 'asserted' refers to a high logic state.

Data Source IP Core:

It is used exclusively to transmit the pre-stored samples of a test sequence (or a speech signal) to our frame blocking IP core at a desired rate, equivalent to the sampling time period (t_s) of a typical analog-to-digital (A2D) converter. The rate is controllable by an end user via a `numClockCycleDelay` input. When this input is loaded with a value, say L_w, the IP core transmits the pre-stored samples with a delay (or latency) of L_w clock cycles, equivalent to the desired sampling time period (t_s), where $L_w = \lfloor t_s/t_c \rfloor$ or $\lfloor f_c/f_s \rfloor$.

In our block design, we have configured our data source IP core to pre-store samples of a short finite-length test sequence, containing 54 unsigned integers, linearly increasing from 1 to 54 and having a word size of $n = 16$ bits. Further, we opt to transmit these pre-stored samples to our frame blocking IP core with a sampling time period of $t_s = 0.0625$ *ms* (for $f_s = 16$ *kHz*), equivalent to a delay (or latency) of $L_w = 3,125$ clock cycles (for $f_c = 50$ *MHz*). The I/O ports of the data source IP core are described in Table S2.

Table S3. I/O ports of the VIO IP core.

Port	Direction	Width (Bits)	Description
`clk`	Input	1	Clock port connected to a single-ended clock source[1].
`probe_out0`	Output	5	Data port, connected to the N_f input port of the frame blocking IP core to load the configuration data, $N_f \in [2, 16]$.
`probe_out1`	Output	4	Data port, connected to the N_o input port of the frame blocking IP core to load the configuration data, $N_o \in [0, N_f - 1]$.
`probe_out2`	Output	14	Data port, connected to the `numClockCycleDelay` input port of the data source IP core to load the configuration data, $L_w \in [1, 16383]$.
`probe_out3`	Output	1	Data port, connected to the `trigger` input port of the pulse generator IP core to load a binary logic.

[1] `clk_out1` output port of the clocking wizard IP core. Note: The end user loads data for ports `probe_out0`, `probe_out1`, `probe_out2` and `probe_out3` via the VIO console from the host computer.

Pulse Generator IP Core:

It generates a single rectangular pulse output (`pulse`) with a width equal to one clock period (t_c) upon detecting a low-to-high logic transition (or raising or positive edge) at its input (`trigger`). The pulse output can be used to trigger the required downstream circuitry. We used the pulse generator IP core in our block design to trigger the data source and ILA IP cores.

Frame Blocking IP Core:

In our block design, we have customized our frame blocking IP core by: setting the 'Word Size' parameter to 16 (as the pre-stored samples of our data source IP core have a word size of $n = 16$ bits) and the 'Frame Size (default)' parameter to 16 (i.e., $N_f = M = 16$ samples that satisfies the practical condition, $N_f < (\lceil f_c/f_s \rceil - \delta)$, of our IP core to perform proper framing, for $f_c = 50\ MHz$ and $f_s = 16\ kHz$); and enabling the optional input N_f and optional outputs `frameSampleIndex` and `ready`. For this IP configuration, the frame size and frame overlap size can be varied as $N_f \in [2, 16]$ and $N_o \in [0, N_f - 1]$, respectively. Therefore, to test and validate the functionality of our IP core for the mentioned configuration, we have considered four test cases, wherein the configuration data (input) and the expected frames (output) from the test sequence for the four test cases are provided in Listings SA2, SA3, SA4 and SA5, presented in Appendix SA.

VIO IP Core:

It is a customizable IP core that can monitor and drive the internal FPGA signals in real-time. The number and width of the I/O ports are customizable in size to interface with the FPGA design. In our block design, we have configured the VIO IP core to drive the input ports of our frame blocking, data source and pulse generator IP cores. The I/O ports of the VIO IP core are described in Table S3.

ILA IP Core:

It is a customizable IP core that can monitor the internal signals of a design. The number and width of the input ports are customizable in size to interface with the FPGA design. In our block design, we have configured the ILA IP core to monitor the output ports of our frame blocking IP core. The input ports of the ILA IP core are described in Table S4.

The block design, illustrated in Figure S1, is finally implemented on our FPGA target (i.e., ZedBoard™) through a series of steps, including design validation, top-level HDL (hardware description language) wrapper creation, synthesis, implementation, bitstream generation and device programming, via a host computer, as illustrated in Figure S2a. The frame blocking IP core is then tested for its functionality using four test cases by:

Table S4. Input ports of the ILA IP core.

Port	Width (Bits)	Type	Description
`clk`	1	Clock	Clock port connected to a single-ended clock source[1].
`probe0`	1	Trigger	The `pulse` output port of the pulse generator IP core drives this port.
`probe1`	1	Trigger	The `frameValid` output port of the frame blocking IP core drives this port.
`probe2`	16	Data	The `frameSample` output port of the frame blocking IP core drives this port.
`probe3`	4	Data	The `frameSampleIndex` output port of the frame blocking IP core drives this port.

[1] `clk_out1` output port of the clocking wizard IP core. Note: `probe0` is configured for trigger setup such that the ILA is triggered when a low-to-high logic transition (or raising or positive edge) is detected at `probe0`. `probe1` is configured for capture setup such that the ILA captures data at `probe2` and `probe3` when a high logic is detected at `probe1`.

launching VIO and ILA consoles on our host computer; providing configuration (L_w), test case (N_f and N_o) and trigger (for data source and ILA IP cores) inputs through the VIO console; and observing the captured outputs (`frameSample` and `frameSampleIndex`) in the ILA console, as illustrated in Figure S2b. We have verified the functionality of our frame blocking IP core for the four test cases by comparing the frames captured in the ILA console with the expected frames provided in Listings SA2, SA3, SA4 and SA5. We found that the four test cases are passed, and our frame blocking IP core is functioning as intended. Further, the inputs fed via the VIO console and the outputs captured in the ILA console regarding the four test cases are illustrated in Figures S3, S4, S5 and S6. Refer to supplementary materials, specifically Video S1, for a comprehensive demonstration of testing and validation of our frame blocking IP core's functionality on an FPGA target.

Abbreviations

The following abbreviations are used in this supplementary materials:

A2D	analog-to-digital
APSoC	all programmable system-on-chip
DUT	design under test
FHA	framing hardware architecture
FPGA	field-programmable gate array
HDL	hardware description language
ILA	integrated logic analyzer
I/O	input and output
IP	intellectual property
JTAG	joint test action group
MMCM	mixed-mode clock manager
SP-RAM	single-port random access memory
USB	universal serial bus
VIO	virtual input/output

Symbols

The following symbols are used in this supplementary materials:

f_c	clock frequency in MHz.
f_s	sampling frequency of an audio signal in kHz.
L_w	latency (in no. of clock cycles) with which the audio signal samples arrive at the input of our FHA (and frame blocking IP core) when the signal is sampled at f_s kHz.
M	depth of SP-RAM.
n	width of SP-RAM; and audio sample word size in bits.
N_f	frame size, denoting the no. of samples in an audio frame.
N_o	frame overlap size, denoting the no. of overlapped samples between adjacent audio frames.
t_c	clock period in ns.
t_s	sampling period of an audio signal in ms.

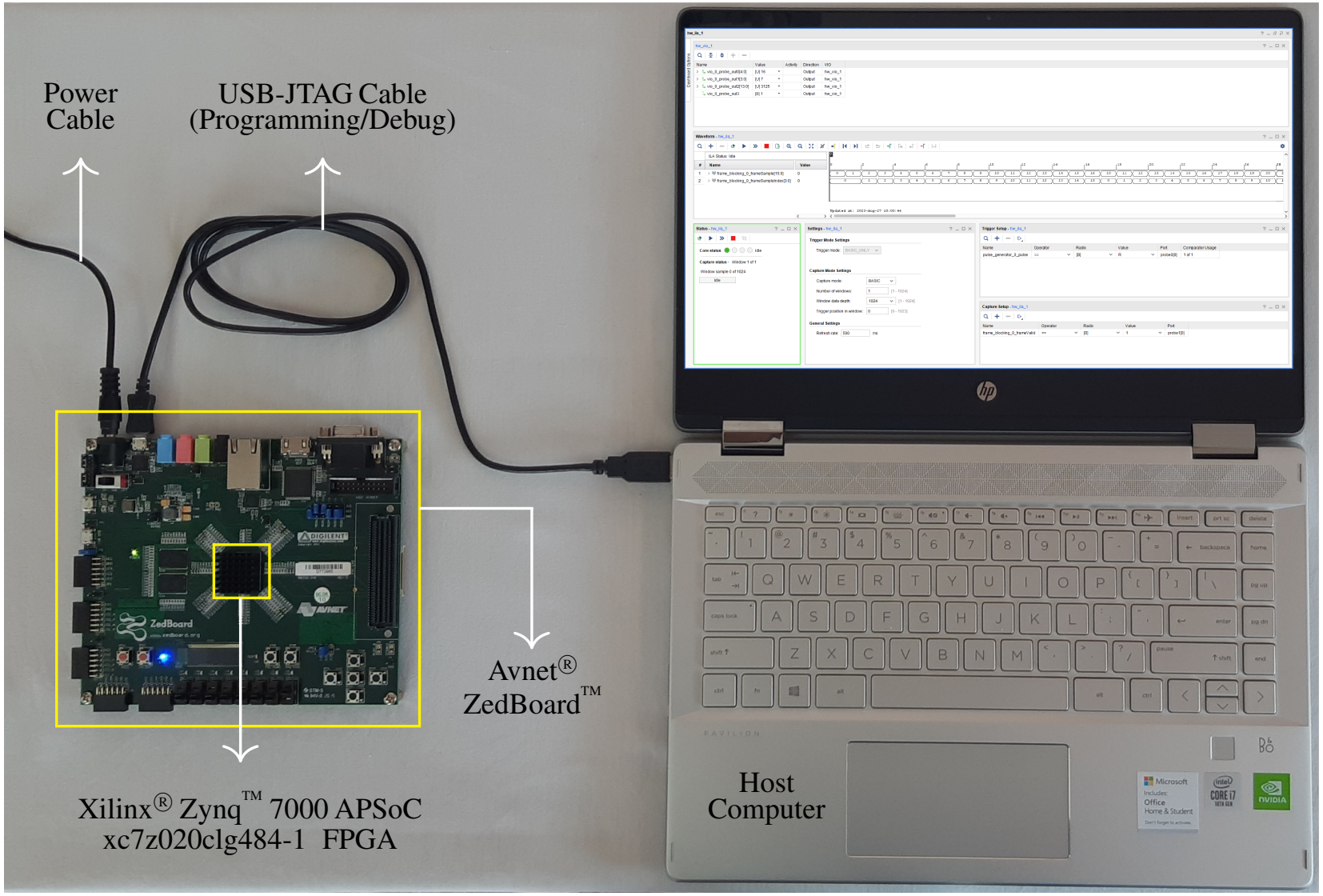

Figure S2. (a) Experimental hardware setup to implement, test and validate the functionality of our proposed frame blocking IP core on an FPGA target.

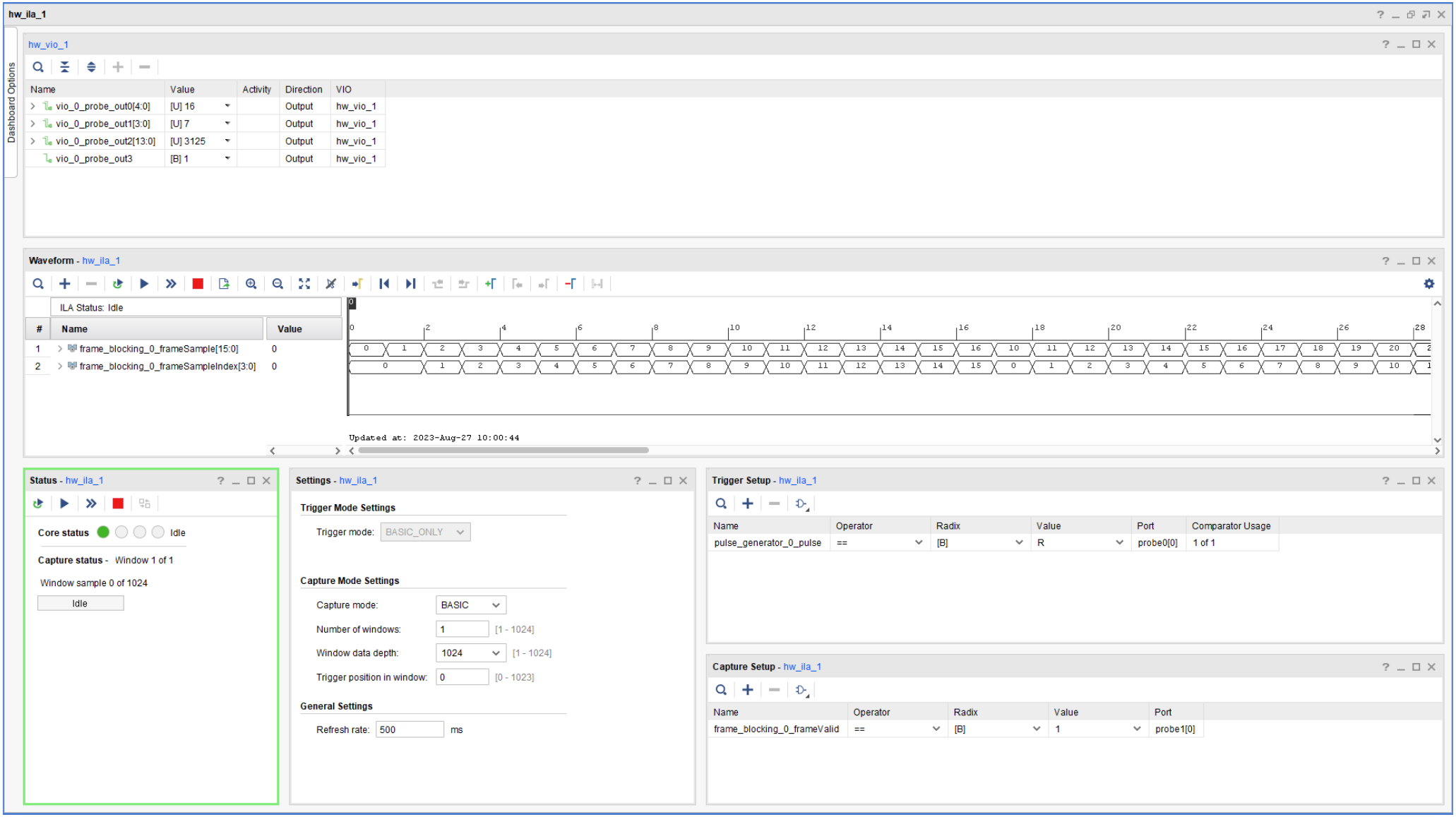

Figure S2. (b) VIO and ILA consoles launched on our host computer.

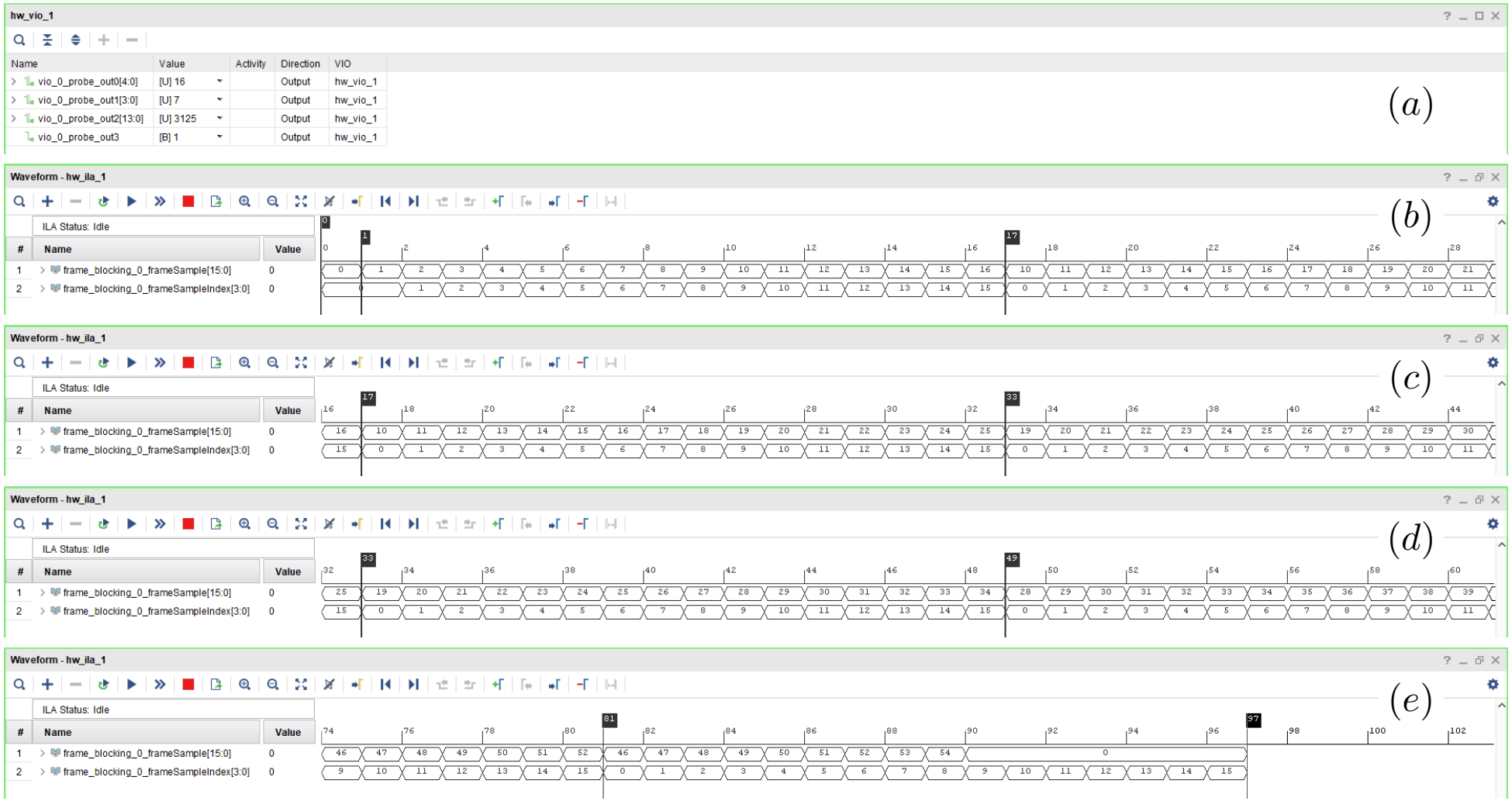

Figure S3. VIO and ILA (waveform) consoles, illustrating the configured inputs and captured outputs, respectively, regarding the first test case (see Listing SA2). **(*a*)** VIO console, illustrating the configured inputs, viz., N_f and N_o (frame blocking IP core), `numClockCycleDelay` (data source IP core) and `trigger` (pulse generator IP core). **(*b*)–(*e*)** ILA (waveform) console, illustrating the captured outputs, viz., `frameSample` and `frameSampleIndex` (frame blocking IP core), highlighting the specific portions corresponding to the first, second, third and last frame, respectively.

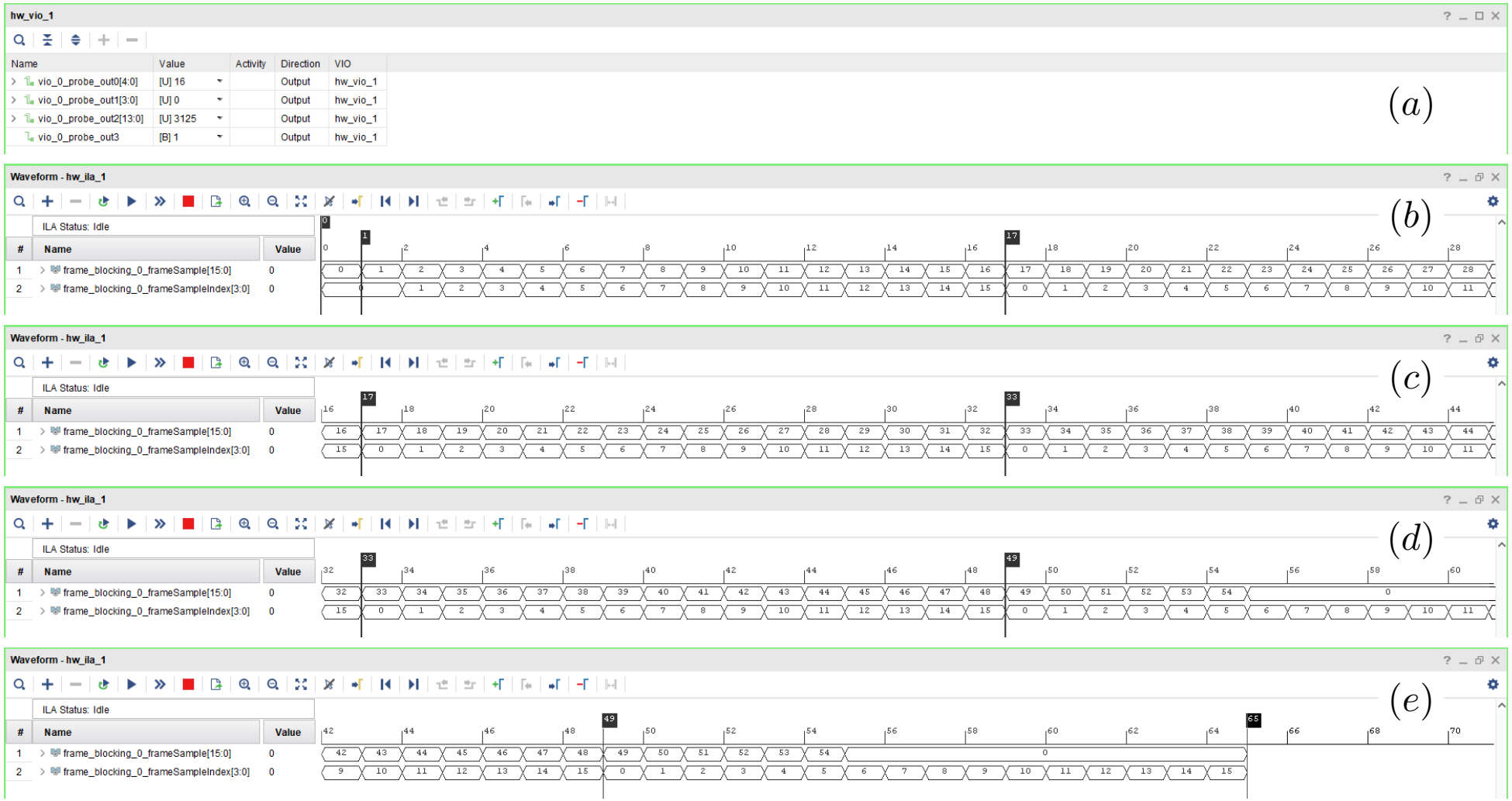

Figure S4. VIO and ILA (waveform) consoles, illustrating the configured inputs and captured outputs, respectively, regarding the second test case (see Listing SA3). **(*a*)** VIO console, illustrating the configured inputs, viz., N_f and N_o (frame blocking IP core), `numClockCycleDelay` (data source IP core) and `trigger` (pulse generator IP core). **(*b*)–(*e*)** ILA (waveform) console, illustrating the captured outputs, viz., `frameSample` and `frameSampleIndex` (frame blocking IP core), highlighting the specific portions corresponding to the first, second, third and last frame, respectively.

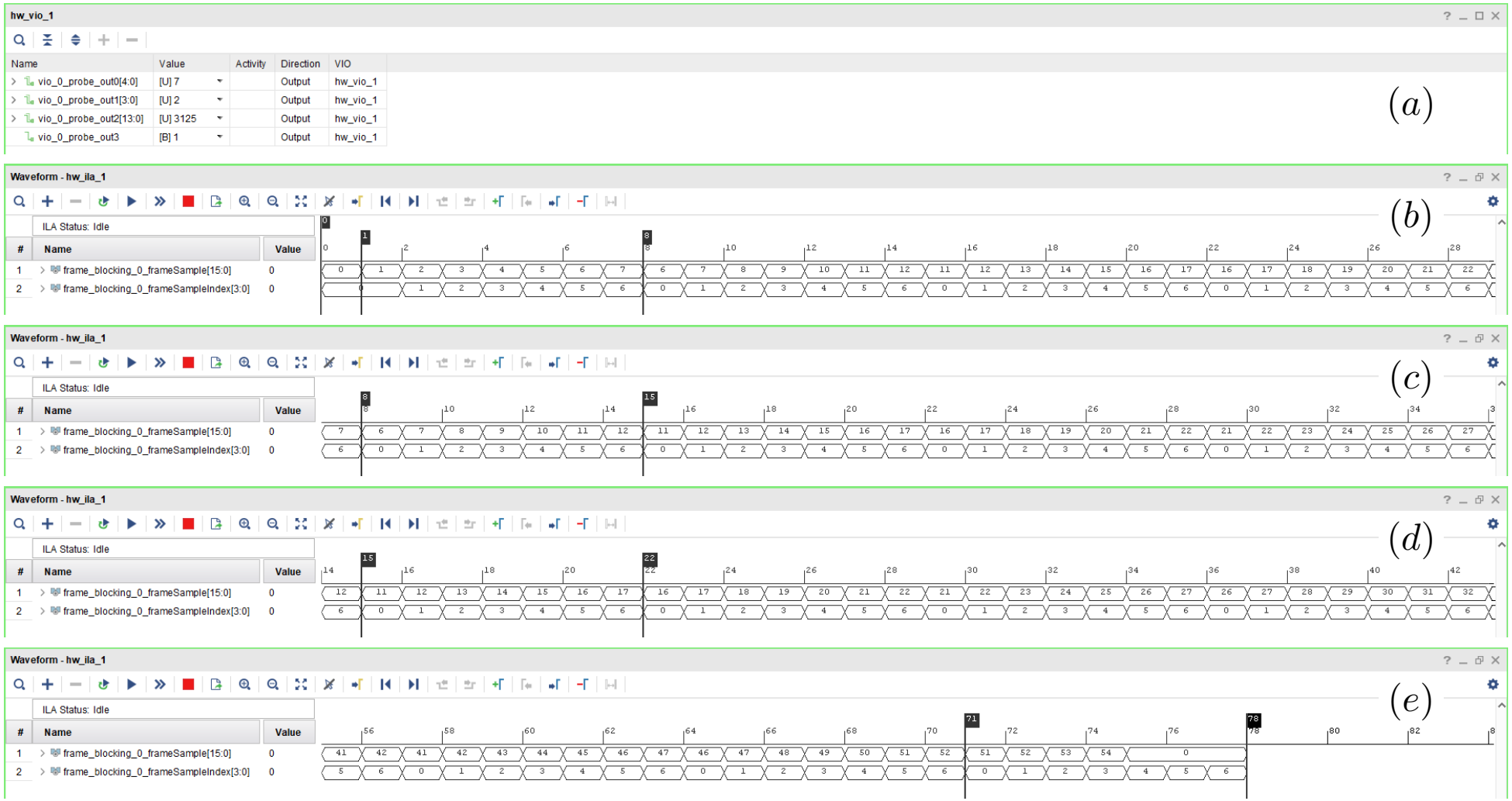

Figure S5. VIO and ILA (waveform) consoles, illustrating the configured inputs and captured outputs, respectively, regarding the third test case (see Listing SA4). **(*a*)** VIO console, illustrating the configured inputs, viz., N_f and N_o (frame blocking IP core), `numClockCycleDelay` (data source IP core) and `trigger` (pulse generator IP core). **(*b*)–(*e*)** ILA (waveform) console, illustrating the captured outputs, viz., `frameSample` and `frameSampleIndex` (frame blocking IP core), highlighting the specific portions corresponding to the first, second, third and last frame, respectively.

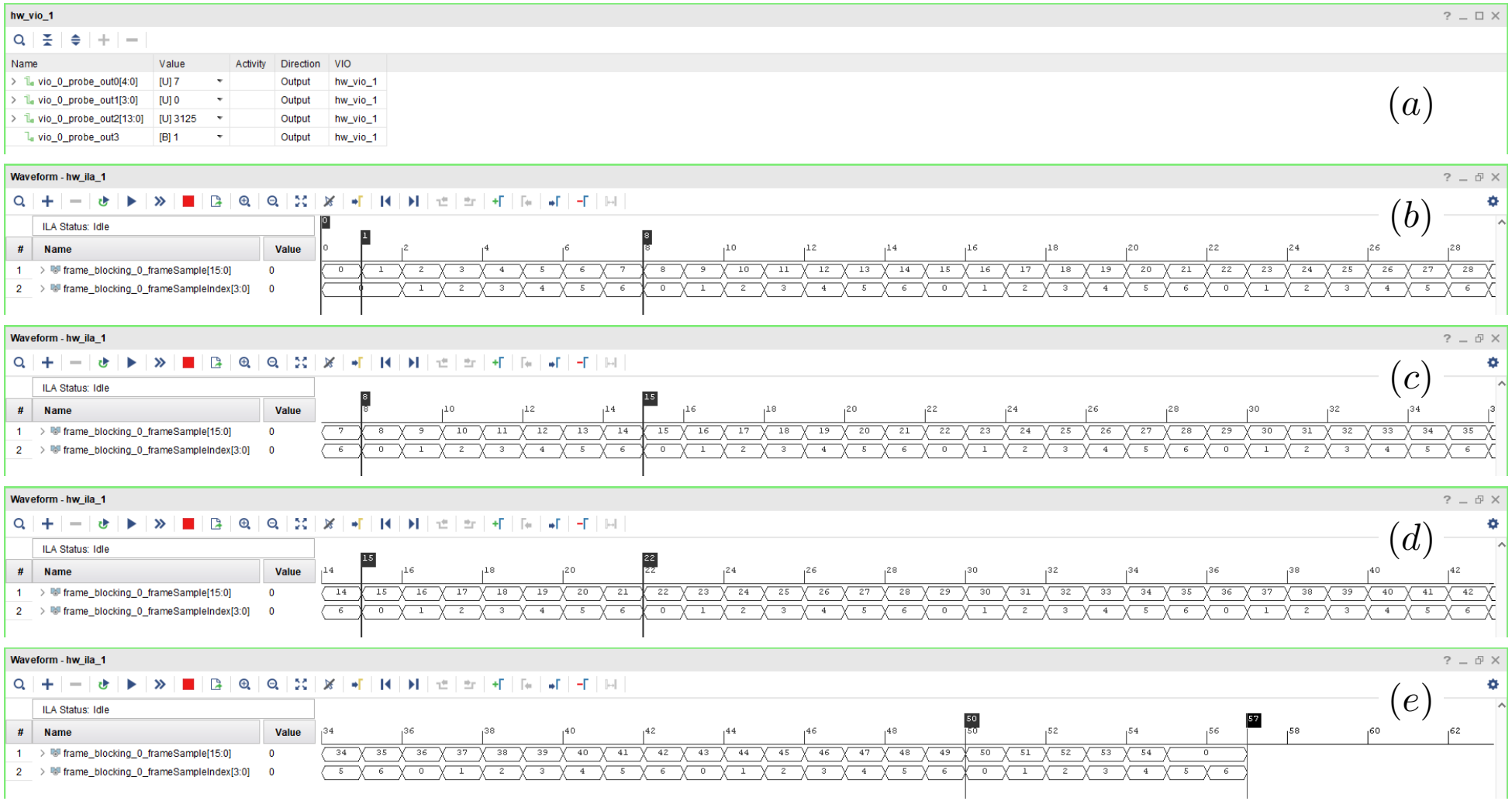

Figure S6. VIO and ILA (waveform) consoles, illustrating the configured inputs and captured outputs, respectively, regarding the fourth test case (see Listing SA5). **(*a*)** VIO console, illustrating the configured inputs, viz., N_f and N_o (frame blocking IP core), `numClockCycleDelay` (data source IP core) and `trigger` (pulse generator IP core). **(*b*)–(*e*)** ILA (waveform) console, illustrating the captured outputs, viz., `frameSample` and `frameSampleIndex` (frame blocking IP core), highlighting the specific portions corresponding to the first, second, third and last frame, respectively.

Appendix SA

Listing SA1. Expected frames (output) for configuration data (inputs) $N_f = 7$ and $N_o = 2$.

Frame-1:	1	2	3	4	5	6	7
Frame-2:	6	7	8	9	10	11	12
Frame-3:	11	12	13	14	0	0	0

Listing SA2. Expected frames (output) for configuration data (inputs) $N_f = 16$ and $N_o = 7$.

Frame-1:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Frame-2:	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Frame-3:	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
Frame-4:	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
Frame-5:	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
Frame-6:	46	47	48	49	50	51	52	53	54	0	0	0	0	0	0	0

Listing SA3. Expected frames (output) for configuration data (inputs) $N_f = 16$ and $N_o = 0$.

Frame-1:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Frame-2:	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Frame-3:	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Frame-4:	49	50	51	52	53	54	0	0	0	0	0	0	0	0	0	0

Listing SA4. Expected frames (output) for configuration data (input) $N_f = 7$ and $N_o = 2$.

Frame-1:	1	2	3	4	5	6	7
Frame-2:	6	7	8	9	10	11	12
Frame-3:	11	12	13	14	15	16	17
Frame-4:	16	17	18	19	20	21	22
Frame-5:	21	22	23	24	25	26	27
Frame-6:	26	27	28	29	30	31	32
Frame-7:	31	32	33	34	35	36	37
Frame-8:	36	37	38	39	40	41	42
Frame-9:	41	42	43	44	45	46	47
Frame-10:	46	47	48	49	50	51	52
Frame-11:	51	52	53	54	0	0	0

Listing SA5. Expected frames (output) for configuration data (input) $N_f = 7$ and $N_o = 0$.

Frame-1:	1	2	3	4	5	6	7
Frame-2:	8	9	10	11	12	13	14
Frame-3:	15	16	17	18	19	20	21
Frame-4:	22	23	24	25	26	27	28
Frame-5:	29	30	31	32	33	34	35
Frame-6:	36	37	38	39	40	41	42
Frame-7:	43	44	45	46	47	48	49
Frame-8:	50	51	52	53	54	0	0

References

1. *ZedBoard*; Available online: https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/ (accessed on 1 April 2024).
2. *Zynq-7000 SoC Data Sheet: Overview (DS190)*; Xilinx Inc., San Jose, California, USA., 2018. Available online: https://docs.amd.com/v/u/en-US/ds190-Zynq-7000-Overview (accessed on 1 April 2024).
3. *Clocking Wizard v6.0, LogiCORE IP Product Guide (PG065)*; Xilinx Inc., San Jose, California, USA, 2022. Available online: https://docs.amd.com/r/en-US/pg065-clk-wiz (accessed on 1 April 2024).
4. *Virtual Input/Output v3.0, LogiCORE IP Product Guide (PG159)*; Xilinx Inc., San Jose, California, USA, 2018. Available online: https://docs.amd.com/v/u/en-US/pg159-vio (accessed on 1 April 2024).
5. *Integrated Logic Analyzer v6.2, LogiCORE IP Product Guide (PG172)*; Xilinx Inc., San Jose, California, USA, 2016. Available online: https://docs.amd.com/v/u/en-US/pg172-ila (accessed on 1 April 2024).