IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 9, 2019, accepted April 3, 2019, date of publication April 11, 2019, date of current version April 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2910391

OpenCL Implementation of FPGA-Based Signal Generation and Measurement

IMAN FIRMANSYAH[1,2], (Student Member, IEEE), AND YOSHIKI YAMAGUCHI[3], (Member, IEEE)

[1]Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba 305-8573, Japan
[2]Indonesian Institute of Sciences (LIPI), Bandung 40135, Indonesia
[3]Faculty of Engineering, Information and Systems, University of Tsukuba, Tsukuba 305-8573, Japan

Corresponding author: Iman Firmansyah (iman.firmansyah@lipi.go.id)

This work was supported in part by the Japan Society for the Promotion of Science (JSPS) Grant-in-Aid for Scientific Research (KAKENHI) under Grant JP17H01707, and in part by the Ministry of Research, Technology and Higher Education (RISTEKDIKTI) of the Republic of Indonesia through the Program Research and Innovation in Science and Technology (RISET-Pro) World Bank Loan under Grant 8245-ID.

ABSTRACT Signal generation and measurement have been widely used in many engineering applications, such as for creating test signals in radar, communication, and software-defined radio. In field programmable gate array (FPGA) design, to generate and measure an analog signal, compatible software and hardware interfaces are required. Traditionally, hardware description language (HDL) is required to program an FPGA. HDL programming provides an efficient logic resource with low latency. However, it is time-consuming for designs that are more complex. Currently, OpenCL is implemented for FPGA programming. OpenCL reduces the FPGA development time because it increases the abstraction level of the code. OpenCL is an open and royalty-free framework for accelerating the algorithm executed on a heterogeneous system, such as a GPU, CPU, DSP, or FPGA. OpenCL implementation on FPGAs yields high-performance results for the computation process. However, compared to HDL design, OpenCL does not provide a particular function to access the FPGA hardware directly. In this paper, we have demonstrated the implementation of OpenCL programming on an FPGA for signal generation and measurement. We have developed OpenCL components that can interact with the FPGA hardware directly. An OpenCL I/O channel extension is employed in the kernel to read data from and write data to the OpenCL components. The experimental results indicate that OpenCL can be used for signal measurement and generation using FPGAs.

INDEX TERMS FPGA, I/O channel, OpenCL, signal generation, signal measurement.

I. INTRODUCTION

Field programmable gate arrays (FPGAs) have been widely implemented in many engineering and scientific applications because of their reusability, reliability, high performance, and low power consumption. FPGAs are used in medical, automotive, and military applications [1], in communications [2], and in nuclear facilities [3]. Recently, FPGAs have also been applied in artificial intelligence (AI) [4] and internet of things (IoT) solutions [5]. FPGAs can also be implemented in signal generation and measurement. This implementation is often applied in digital signal processing [6], data acquisition [7], communication [8], software-defined radio [9], automotive radar [10], and quantum computing [11].

Traditionally, an FPGA is programmed using hardware description language (HDL) to generate hardware implementation from the source code onto a register transfer level (RTL). However, owing to an increase in design complexity, FPGA programming using HDL is time-consuming. Moreover, to develop a complex design using HDL, an FPGA programmer needs to have detailed knowledge of the hardware and software of an FPGA, particularly its programming, simulation, and debugging process.

Currently, high-level synthesis (HLS) is implemented on FPGAs. HLS provides an alternative solution to reduce the development time for FPGA programming. HLS improves the FPGA design efficiency by increasing the abstraction level of the code [12]. HLS also reduces the gap between the FPGA design and the programming process. Consequently, the FPGA development time can be reduced. Various research studies present the implementation of HLS in FPGA projects,

The associate editor coordinating the review of this manuscript and approving it for publication was Jing Liang.

particularly in the high-performance computing applications. In [13], FPGAs are employed for molecular dynamics simulation using Intel's OpenCL SDK. In [14], tsunami simulations are performed on FPGAs using OpenCL. In [15], FPGAs are used to compute high-performance stencils by combining spatial and temporal blockings using OpenCL. In [16], OpenCL implementations for high-performance computing application using FPGAs are demonstrated to be more energy-efficient than those using GPUs. Further, in [17], SDAccel and Vivado HLS are used for data mining using an FPGA accelerator.

In this paper, we focus on the implementation of FPGAs for signal generation and measurement. Therefore, an FPGA needs to access external devices such as an analog-to-digital converter (ADC) and a digital-to-analog converter (DAC) for measuring and generating a signal. To reduce the development time, OpenCL was selected to program the FPGA because OpenCL exploits the concept of parallelism that enables us to develop a parallel program application for FPGAs using high-level language. In addition to this, OpenCL avoids creating complex HDL codes, particularly for the libraries as well as platform-specific tools [18]. However, we faced problems in developing the FPGA-based signal measurement and generation using OpenCL. Compared to the HDL program, OpenCL does not provide direct access to the FPGA's I/O, particularly for reading data from an ADC and writing data to a DAC.

To overcome these limitations, we developed ADC and DAC component modules on the *system.qsys* of the FPGA's board support package (BSP), which allows an OpenCL kernel to access the FPGA's I/O. To enable the kernel to communicate with the ADC and DAC components, an OpenCL I/O channel extension was employed. This channel extension allowed the OpenCL kernel to stream data to and from the FPGA's I/O. Therefore, this study demonstrates the capability of the OpenCL program for accessing the FPGA's I/O directly, particularly for signal measurement and generation applications. It is expected that this research will contribute to the use of the OpenCL program not only for FPGA-based parallel computations, but also for signal and video processing, data acquisition, and control systems through the FPGA's I/O.

Here, we present several advantages of using OpenCL implementation for the FPGA-based signal generation and measurement compared to using HDL-based design. In HDL implementation, to generate a signal, a ROM-based lookup-table is required to store data to generate a signal. The size of data is also limited by the size of the FPGA's ROM. In some cases, the FPGA needs to be reprogrammed when different signals need to be modified. However, for an OpenCL implementation, the signal data can be stored on global memory (external DDR memory) instead of the FPGA's ROM because the OpenCL framework provides an interfaces and access to the global memory. In this implementation, large data can be stored on global memory where this data is limited by the size of the external memory. Different signals can also be updated quickly without reprogramming the FPGA by invoking the host to transmit the data to the FPGA's global memory. Similarly, the measured signal can also be stored on global memory. Consequently, this allows the host to read the data from the FPGA directly for further processing and analysis. In HDL-based design, these implementations require detailed specifications of the double data rate (DDR) interface for the configuration for the DDR memory controller to access external memory. Moreover, the FPGA simulation and debugging process needs to be performed. To enable communication or to transmit and receive the data between the FPGA and the host, hardware configuration and a device driver for a PCIe hard IP or a 10 Gbps Ethernet controller is required in the HDL-based design. However, for an OpenCL implementation, the PCIe and Ethernet controller are generated automatically. This is because the OpenCL framework consists of firmware, software and device driver between FPGA and the host for connecting, controlling and transferring data [19]. In term of development time, OpenCL implementation takes two weeks of programming the FPGA, particularly for signal measurement and generation, where the most considerable portion involves developing the ADC and DAC component modules using Avalon-ST source and Avalon-ST sink on the FPGA's BSP. Thus, OpenCL implementation reduces the development time and increases productivity.

This paper presents OpenCL kernel implementation for signal measurement and generation. To evaluate the OpenCL kernel implementation within this context, we conduct experiments by developing the kernel into three categories, as follow:

- *signal measurement*: In this implementation, the OpenCL kernel measures the input signal and stores the data in global memory so that the host can read the data for further analysis. The example implementation of this kernel involves data acquisition from sensors as well as signal filtering.
- *signal generation*: Here, the host writes the data to global memory, and then the OpenCL kernel generates an output signal by reading the data from global memory. An example implementation of this kernel includes signal generation for wireless communication or for a radar transmitter.
- *signal measurement and generation*: In this implementation, the first kernel reads a signal and writes it to an I/O channel. The second kernel reads the data from the I/O channel and generates a signal simultaneously. Here, data transfer is performed without accessing global memory. An example of this implementation involves digital signal processing.

The remainder of this paper is organized into six additional sections. In Section 2, we present some related works. In Section 3, we introduce the OpenCL implementation for FPGA. In Section 4, the customization of the FPGA hardware for developing the OpenCL components (ADC and DAC) is described. Section 5 presents the OpenCL system implemen-

tation and the experimental results. In Section 6, discussions are provided. Finally, the study is summarized in Section 7.

II. RELATED WORK

This section presents some related studies on the implementation of FPGA-based signal generation and measurement in many applications. In [20], an FPGA-based signal generator is developed on Altera's Cyclone II based on direct digital synthesis (DDS) using VHDL to generate a sine wave, square wave, triangular wave, saw-tooth wave, ladder wave, and amplitude modulated wave. In [21], it is demonstrated that a DDS-based signal generator that uses an FPGA by modifying and improving the pipelined accumulator with a CORDIC algorithm can solve problems caused by the usage of traditional DDS. Similarly, the same method is implemented in [22]. The paper presents the design implementation of a high-speed arbitrary signal generator on a Xilinx Spartan 6 FPGA based on DDS technology and a high-speed AD9734 DAC. In [23], a signal generator is also implemented for FPGA-based nuclear magnetic resonance (NMR) spectroscopy on an Altera Cyclone II FPGA using DDS and VHDL. Another implementation of an FPGA-based signal generator can be found in [24]. The paper presents the use of an FPGA for an arterial pulse wave generator using an Altera Cyclone FPGA based on the Nios II embedded processor. In [25], it is demonstrated that an FPGA-based signal generator for 4-level pulse amplitude modulation (PAM-4) signaling can control the AD9739A DAC using a Xilinx Kintex-7 FPGA. The papers mentioned above present the implementation of an FPGA-based signal generator that uses Verilog/VHDL for the hardware development. For storing the signal to be generated, an FPGA's ROM is required as a look-up table for the design.

The implementation of an FPGA-based signal measurement can also be found in [23]. In this paper, a signal acquisition to process the digital quadrature demodulation in nuclear magnetic resonance (NMR) spectroscopy using VHDL on an Altera Cyclone II FPGA, has been demonstrated. In [26], a data acquisition (DAQ) system, which is designed to determine plasma electron densities using a Xilinx Kintex-7 FPGA and VHDL code, has been presented. A signal measurement is also implemented in software-defined radio. In [27], an example of an FPGA's implementation as a digital front-end module that receives digital signals that are converted by the ADC from the IF signals, has been described. The use of OpenCL programming for data acquisition using FPGAs is mentioned in [28]. However, the latter study does not provide a comprehensive and detailed analysis of the results, particularly results pertaining to signal generation and measurement. According to the study, as presented above, many applications employ signal generation and measurement using an FPGA in their design. Therefore, this motivates us to focus on this study by implementing OpenCL for signal generation and measurement using FPGAs.

III. OPENCL FOR FPGA

In this section, we briefly introduce OpenCL for an FPGA including the memory type. In OpenCL programming of an FPGA, the kernel code which is executed by the FPGA has the same programming model as the code of a graphics processing unit (GPU). However, the kernel code for the FPGA is compiled into a different sequence of instructions that are executed by different work items simultaneously. This is because an FPGA exploits pipeline parallelism to execute the kernel. For high-performance computing purposes, OpenCL provides an application program interface (API) that allows the host to communicate with the FPGA through a PCIe bus. For embedded purposes, such as use of an SoC FPGA, the internal bus is used for data transfer and communication between an FPGA and the advanced RISC machine (ARM) processors, as shown in Fig.1.

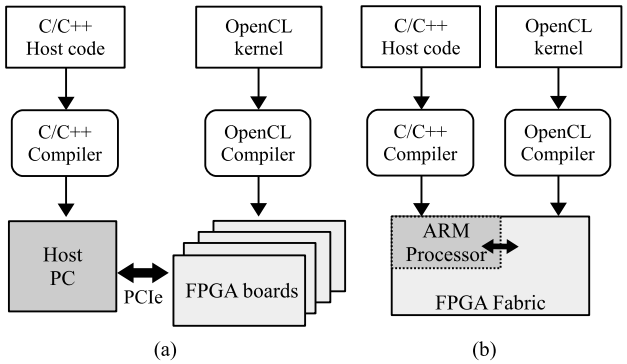

FIGURE 1. OpenCL system with a host CPU and FPGAs (a) data communication through a PCIe, (b) using the internal bus.

OpenCL for FPGAs also shares the same memory types for the computation process. The memory types are defined as global/constant memory, local memory, and private memory. The offline compiler for OpenCL employs DDR3/DDR4 memory on an FPGA board as global memory. The memory type that has higher throughput with lower latency is local memory. During the kernel compilation, local memory is implemented by the block RAMs. This memory is dedicated to work items in the same workgroup. The last type of memory that has faster throughput and smaller size than the others is the private memory. Depending on data size, private memory is implemented by either block RAMs or registers [29].

IV. CUSTOMIZING THE BOARD HARDWARE FOR OPENCL COMPONENTS

In this section, we demonstrate how to develop the ADC and DAC component modules on the FPGA's board support package (BSP), which allow the OpenCL kernel to access the FPGA's I/O. Then, the ADC and DAC component attributes are defined in the channel interface of the *board_spec.xml* file that describes the hardware interfaces to the Intel FPGA SDK for OpenCL. Finally, the use of the OpenCL I/O channel

extension is explained for streaming data to and from an FPGA's I/O through the ADC and DAC components.

A. DEVELOPING OPENCL'S ADC AND DAC COMPONENTS

Fig.2 shows a *system.qsys* diagram of the FPGA's BSP where the OpenCL ADC and DAC components have been developed. The BSP, which is provided by the FPGA vendor for OpenCL programming, simplifies FPGA programming because it provides an external DDR memory controller for writing and reading data to global memory, and provides a data interface for communications between the host and FPGA.

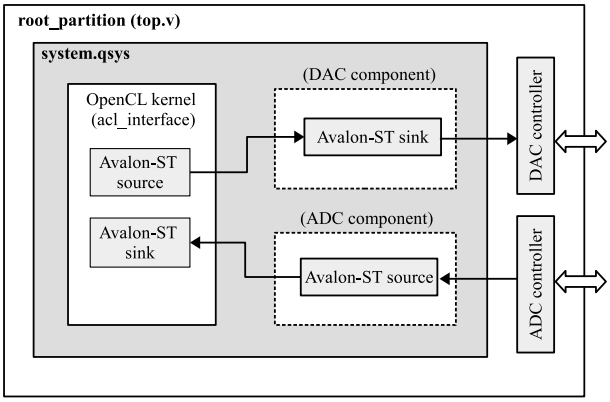

FIGURE 2. System Qsys of a customized board support package (BSP) by adding new ADC and DAC components.

The first component is the ADC component. This component is created by implementing an Avalon-ST source for streaming data from the external ADC board to the OpenCL kernel. The OpenCL kernel receives the data from this component through an Avalon-ST sink. The block diagram of the ADC component is shown in Fig.3(a). The ADC component receives two input signals, the clock (*kernel_clk*) and the reset (*kernel_rst*) from the kernel, has a port for reading the data from the ADC board (*adc_read*), produces two signals for handshaking with the kernel: (*kernel_ready*) and (*kernel_valid*), and includes a port for streaming the data to the kernel (*kernel_out*).

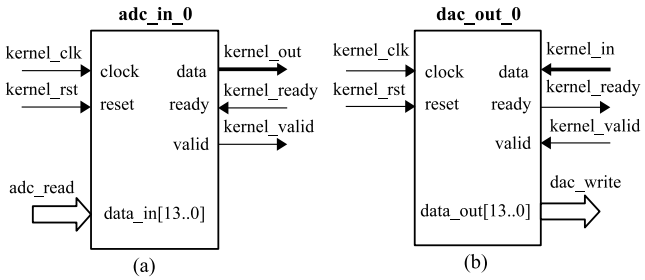

FIGURE 3. (a) ADC component using Avalon-ST source, (b) DAC component using Avalon-ST sink.

The second component is the DAC component. This component is created by implementing an Avalon-ST sink for streaming data from the OpenCL kernel to the external DAC board. The OpenCL kernel streams the data through an Avalon-ST source to the DAC component. As shown in Fig.3(b), this component also has the same signals as the ADC; however, the two signals for handshaking, (*kernel_ready*) and (*kernel_valid*), are in the opposite directions. This component also has a port for receiving streamed data from the kernel (*kernel_in*) and a port for writing data to the external DAC board (*data_out*).

In the Qsys system design, the data ports of the ADC and DAC components need to be exported so that the kernel can read from and write data to the external ADC/DAC board. The *adc_read* port of the ADC component is exported and connected to the data port of the ADC controller, while the *dac_write* port of the DAC component is exported and connected to the data port of the DAC controller. The ADC and DAC controllers are written in HDL and are located inside the root partition (*top.v*) of the BSP to control the ADC/DAC board. The ADC chip on the ADC/DAC board converts an analog signal to a 14-bit digital signal. In contrast, the DAC chip converts a 14-bit digital signal to an analog signal.

B. SETTING OPENCL COMPONENT PARAMETERS

We have shown how to implement the OpenCL's ADC and DAC components on the FPGA's BSP. To allow the OpenCL kernel to access these components, the component attributes need to be declared in the *board_spec.xml* file of the BSP. The *board_spec.xml* file is an extensible markup language (XML) file that provides the board description, such as the hardware interface and the component interface, to the Intel SDK for OpenCL. According to the content of the *board_spec.xml* file, a custom circuit for an FPGA is generated by the SDK compiler for OpenCL. Then, this custom circuit is incorporated with the OpenCL kernel [30].

TABLE 1. OpenCL component attributes.

Component attributes	ADC component	DAC component
name	adc_in_0	dac_out_0
port	adc_read	dac_write
type	streamsource	streamsink
width	14	14
chan_id	ch_adc_read	ch_dac_write

According to the ADC and DAC components, as shown in Fig.3(a) and Fig.3(b), we specify the component attributes such as *name*, *port*, *type*, *width*, and *chan_id* on the channel interface of the *board_spec.xml* file, as shown in Table 1. The *name* attribute specifies the names of the ADC and DAC components. The *port* attribute specifies the data ports of the ADC and DAC components where data are read from and written to the FPGA's I/O. The *type* attribute specifies the type of Avalon-ST bus being used. Because the ADC component reads data from the ADC board and streams this data to the OpenCL kernel, a stream source is employed. On the other hand, the DAC component receives streamed data from the OpenCL kernel and writes the data to the

DAC board; therefore, a stream sink is used. The *chan_id* attribute is a unique name for the I/O interface on the FPGA board and will be associated to the *io*("*chan_id*") attribute in the OpenCL kernel. The value of 14 in the *width* attribute specifies the 14-bit resolution of the ADC and DAC board. In the experiments, the data type for this channel is specified as *ushort*.

C. ACCESSING OPENCL'S ADC AND DAC COMPONENTS USING AN I/O CHANNEL EXTENSION

So far, we have shown how to develop the ADC and DAC components for accessing the FPGA's I/O in the *system.qsys* of the BSP and how to set the components attributes in the *board_spec.xml* file. An OpenCL API call is required to stream data between the OpenCL kernel and the FPGA's I/O across the ADC and DAC components. In this subsection, we introduce the concept of the OpenCL I/O channel extension and how to use the channel extension in our design.

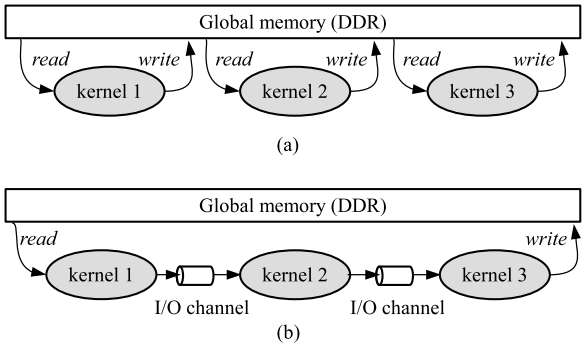

FIGURE 4. Kernel-to-kernel communication (a) without I/O channel through global memory (b) with I/O channel implementation.

In OpenCL design, a kernel needs to communicate with global memory to read and write data for the computation process. When two or more kernels are executed to solve computational problems, more communications and data transfers are performed between the kernels and global memory. Compared to a GPU that supports a high-bandwidth global memory, most FPGA boards are equipped with DDR3 or DDR4 as the global memory. As a result, this causes a reduction in performance owing to the global memory bandwidth bottleneck, as shown in Fig.4(a). To overcome this constraint, an OpenCL I/O channel extension is employed to transfer data among the kernels without accessing global memory, as shown in Fig.4(b). The I/O channel extension is a first-in-first-out (FIFO) buffer. The I/O channel is implemented using RAM blocks and registers [31] [32]. In the Intel SDK for OpenCL, a *write_channel_intel*(*ch_0*, *input_buf*) API call is used to write data to the *input_buf* variable of a channel *ch_0*. To read data from the *ch_1* channel to an *output_buf* variable, a *output_buf* = *read_channel_intel*(*ch_1*) API call is used. Previous study has shown the implementation of an OpenCL channel extension for data communication. In [33], an implementation of the OpenCL I/O channel extension for data communication using a high-speed FPGA network through the QSFP+ port was demonstrated.

In this study, the OpenCL I/O channel extension is employed to stream data between the OpenCL kernel and the FPGA's I/O through the ADC and DAC components. To read a signal from the ADC board, the channel attribute in the kernel must point to the *chan_id* name of the ADC component. Here, the *chan_id* attribute is specified as *ch_adc_read*. Therefore, the channel attribute in the kernel is declared as *io*("*ch_adc_read*"). A similar method is applied to write data to the DAC board. However, the channel attribute in the kernel is declared as *io*("*ch_dac_write*") so that it points to the DAC component.

V. SYSTEM IMPLEMENTATION

In this study, we evaluate the OpenCL kernel using a Cyclone V SoC FPGA board from Terasic. This board consists of the Cyclone V SoC 5CSEMA5F31C6 FPGA, dual-core ARM Cortex-A9 (HPS), 85K programmable logic elements, 4,450 Kbits embedded memory, two 40-pin expansion headers, and two hard memory controllers [34]. For the ADC and DAC chip, we utilized the analog-to-digital/digital-to-analog (AD/DA) board from Terasic. This AD/DA board consists of dual AD channels with 14-bit resolution and dual DA channels with 14-bit resolution [35]. In the experiments, the AD/DA board was connected to the GPIO JP1 and JP2 pins of the Cyclone V DE1-SoC FPGA board. The analog input signal was connected to AD channel A, while the analog output signal was generated from DA channel A. To execute an OpenCL project, Intel SDK for OpenCL version 17.0 was used to compile the kernel. The ARM part of the SoC FPGA executed the host program, which was cross-compiled by Intel's SoC EDS.

A. SIGNAL MEASUREMENT

In this experiment, we demonstrate how to develop the OpenCL kernel for signal measurements using the OpenCL ADC component and a channel extension. The example implementation of this kernel is for data acquisition from sensors or signal processing in a radar receiver.

1) EXPERIMENTAL DESIGN

Fig.5 shows the FPGA-based system design for signal measurement using OpenCL. The analog signal, which is converted to digital by the ADC chip, passes through an I/O channel extension. Then, the OpenCL kernel reads and stores the data in the global memory of the FPGA. The global memory is also accessible by the host, which allows the host to read the data for further analysis. To measure the signal, the kernel attributes are declared according to the content of the *board_spec.xml* file. In the experiment, the name for the *chan_id* attribute was specified as "*ch_adc_read*". Therefore, the channel attribute in the kernel was declared as *__attribute_*((*io*("*ch_adc_read*"))). The *max_global_work_dim*(0) attribute was used to inform the

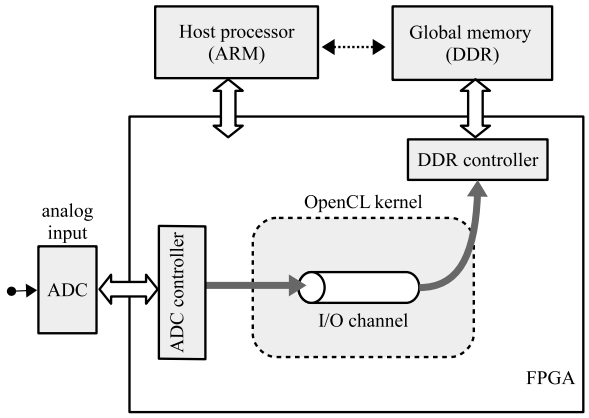

FIGURE 5. I/O channel implementation for signal measurement from ADC to global memory of an FPGA.

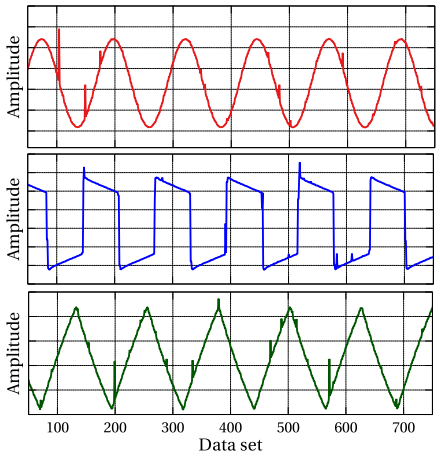

FIGURE 6. Measured input signal by the FPGA from an arbitrary signal generator.

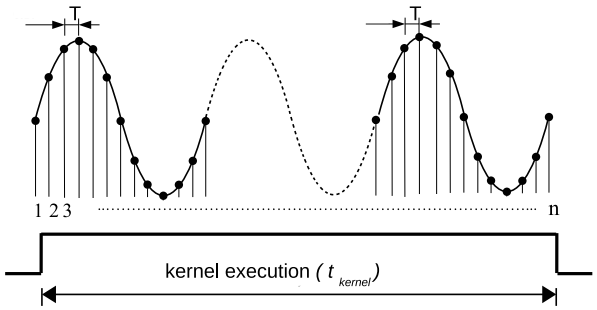

FIGURE 7. Kernel execution time for signal measurement.

OpenCL offline compiler that the kernel type was the single work item kernel.

To read the signal from the channel, the *data_in*[*i*] = *read_channel_intel*(*ch_data_read*) API call was used, where the *ch_data_read* was the name of the channel variable. The results were stored in a *datain* buffer on the global memory of the FPGA. To execute this kernel, the host invoked the *clEnqueueTask*() function in the host code. Meanwhile, the *clEnqueuereadBuffer*() function was called to read the *data_in* data in global memory. The OpenCL kernel for the signal measurement is shown in Listing 1.

```
// adc_channel_read.cl
#pragma OPENCL EXTENSION cl_intel_channels : enable

channel ushort ch_data_read
__attribute__((depth(0)))
__attribute__((io("ch_adc_read")));

__attribute__((max_global_work_dim(0)))
__kernel void adc_channel(__global ushort *restrict
    datain, int length)
{
  for(int i=0; i<length; i++){
      datain[i] = read_channel_intel(ch_data_read);
  }
}
```

Listing 1. OpenCL kernel for signal measurement.

2) IMPLEMENTATION AND RESULT

In this first experiment, we employed an arbitrary signal generator to generate different signal types, such as sine, triangle, and square wave signals. Fig.6 shows examples of the measured input signal types by the OpenCL kernel (sine wave, triangle wave, and square wave). For the remainder of this paper, evaluation of sine waves as both input and output signals will be discussed.

We have shown how to read a signal by leveraging the I/O channel extension in the OpenCL kernel. However, the frequency of the signal still cannot be determined. To evaluate the frequency of the signal, the kernel sampling rate for the measured signal (T) is required. This sampling rate can be calculated from the kernel execution time (t_{kernel}) divided by the length of the data (n), as defined by Equation 1. Fig.7 shows the signal, which is sampled every T seconds over length n of the dataset.

$$T = \frac{t_{kernel}}{n} \tag{1}$$

In the experiment, the signal generator was set to generate a sine wave (f_{in}) with frequency of 20 MHz. The kernel was programmed to read and store the data to global memory with different lengths (n) as follows: 50K, 60K, and 75K. From the experimental results, the kernel execution time was $t_{kernel\ 1} = 0.686\ s$, $t_{kernel\ 2} = 0.801\ s$, and $t_{kernel\ 3} = 0.974\ s$ for $n =$ 50K, $n = 60$K, and $n = 75$K, respectively. By applying the fast Fourier transform (FFT) function, the frequency of the measured signals for different lengths (n) of the dataset are shown in Fig.8. The measured frequencies for the 20 MHz input frequency were 16.85 MHz, 17.31 MHz, and 17.8 MHz for $n = 50$K, $n = 60$K, and $n = 75$K, respectively. The results show that the measured frequency is lower than the input frequency. This is because of the slow kernel execution time owing to the use of global memory for storing the measured signal.

To avoid the global memory constraint, the experiment was carried out by storing the data temporarily in the on-chip

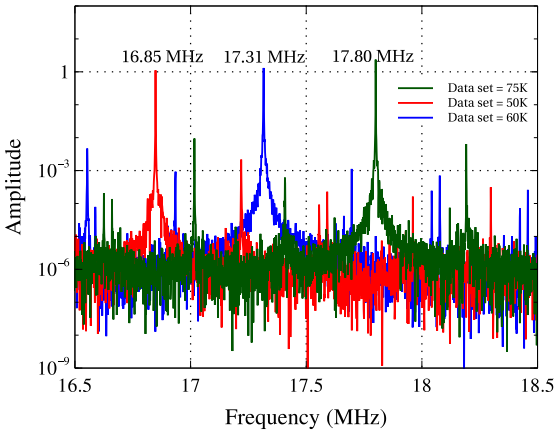

FIGURE 8. Measured frequency using global memory for a 20 MHz frequency input signal.

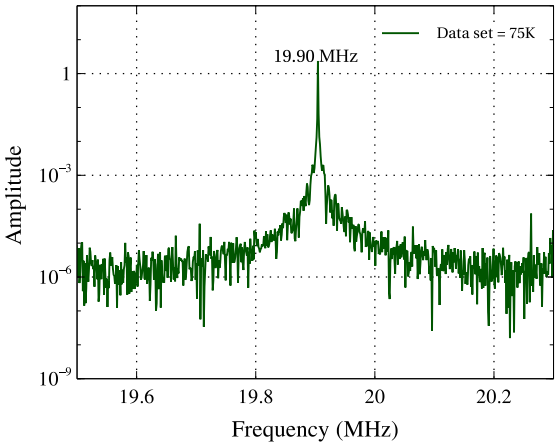

FIGURE 9. Measured frequency using on-chip RAM for a 20 MHz frequency input signal.

RAM of the FPGA instead of writing directly to global memory. The kernel execution time decreased to $t_{kernel3} = 0.871\ s$ for $n = 75$K. The result indicates that the use of an on-chip RAM achieves faster execution time. By using Equation 1 to calculate the kernel sampling rate, the measured frequency input can be evaluated using an FFT function, with the result shown in Fig.9. The measured frequency of 19.9 MHz is closer to the 20 MHz frequency of the input signal than the 17.8 MHz signal measured using global memory access.

B. SIGNAL GENERATION

In this experiment, we demonstrate how to implement the OpenCL kernel for signal generation using the OpenCL DAC component and a channel extension. The example implementation of this kernel applies to signal generation for wireless communication or for a radar transmitter.

1) EXPERIMENTAL DESIGN

To generate an output signal, first the host writes one cycle of a sine wave to the global memory of the FPGA by calling the *clEnqueueWriteBuffer*() function. Second, the host invokes

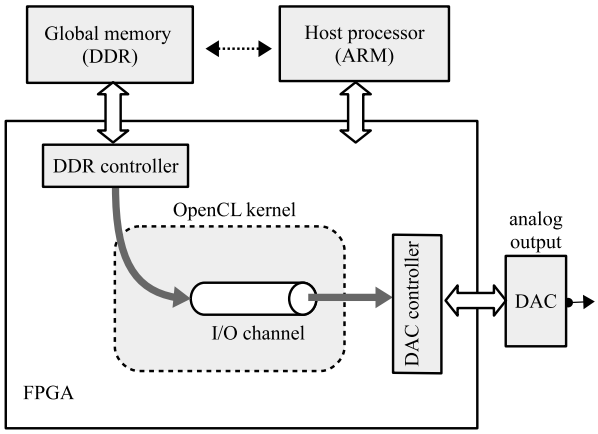

FIGURE 10. I/O channel implementation for signal generation.

the *clEnqueueTask*() function to execute the OpenCL kernel. On the FPGA side, the kernel reads the data from global memory and passes it to the DAC board through an I/O channel extension. This process is depicted in Fig.10. According to the content of the *board_sprec.xml* file, we specified the *chan_id* value as "*ch_dac_write*". Therefore, the kernel attribute was declared as *__attribute__((io("ch_dac_write")))*. To read the data from global memory and to pass it to the DAC board, a *write_channel_intel*(*ch_data_write*, *sine*[*i*]) API call was executed, where *ch_data_write* was the name of the channel variable, and *sine* represented variable arrays containing one cycle of a sine wave. The OpenCL kernel for generating the signal is given in Listing 2. In the kernel, the *length* variable is defined as the length of the data in one cycle. Fig.11 shows how one cycle of a sine wave with length *m* and amplitude from 0 to 2^{14} is stored in the global memory of the FPGA.

```
// dac_channel_write.cl
#pragma OPENCL EXTENSION cl_intel_channels : enable

channel ushort ch_data_write
__attribute__((depth(0)))
__attribute__((io("ch_dac_write")));

__attribute__((max_global_work_dim(0)))
__kernel void dac_channel(__global ushort *restrict
    sine, int length)
{
  while(1){
    for (int i=0; i< length; i++){
      write_channel_intel(ch_data_write, sine[i]);
    }
  }
}
```

Listing 2. OpenCL kernel for signal generation.

2) IMPLEMENTATION AND RESULTS

To evaluate the kernel, an oscilloscope was employed to measure the output signal from the DAC chip. After executing the kernel, the analog output signal was generated as shown in Fig.12. As can be seen, there is a delay after one cycle of the generated signal (red circle), as shown in Fig.12(a).

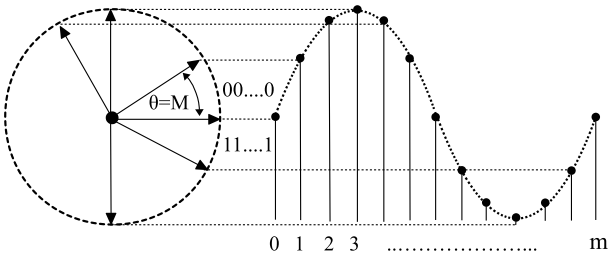

FIGURE 11. One cycle of a sine wave for a dataset of length *m*.

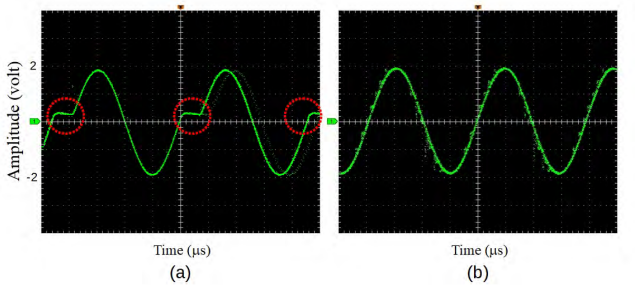

FIGURE 12. Output analog signal (a) with global memory, (b) without global memory implementation.

This delay is owing to the long latency involved in global memory usage. After one period of the signal, the kernel accesses the non-contiguous memory allocation in the global memory and generates the same signal repeatedly. To overcome this limitation, global memory usage is omitted by copying the data from global memory to the on-chip RAM of the FPGA. Fig.12(b) shows the result when the kernel employs the on-chip RAM, indicating that the signal is generated without delay.

We have shown how to generate a signal by leveraging an OpenCL I/O channel extension without any delays. To generate the signal at a specific frequency, we propose a formula which is similar to the direct digital synthesis (DDS) architecture. The DDS technique is used to generate a sinusoidal signal or arbitrary waveform with a programmable frequency. DDS enables us to control the frequency of the signal accurately and to adjust the frequency quickly [36]. A typical DDS architecture consists of a phase accumulator (M), a reference clock f_c, and a DAC. The phase accumulator specifies the phase angle of the output signal. This phase accumulator has N-bit resolution, with a range from 1 to 2^N. The DAC converts the digital value to an analog signal [37] [38], as shown in Fig.11.

In the experiment, we employed the ADC and DAC with 14-bit resolution. Therefore, the dataset for one cycle of the sine wave is represented by the data from 0 to 2^{14}. To generate a signal with a specific frequency, the host writes one cycle of a sine wave with the length of the data (m) to global memory, as shown in Fig.11. Here, we present the equation for estimating the frequency of the generated signal. In DDS, the frequency of the signal with $N - bit$ resolution can be calculated using Equation 2.

$$f_o = M \times \frac{f_c}{2^N} \tag{2}$$

Because m is equal to 2^N divided by M, the frequency of the output signal can be estimated using Equation 3 as follows:

$$F_{estimation} = \frac{f_{kernel}}{m} \tag{3}$$

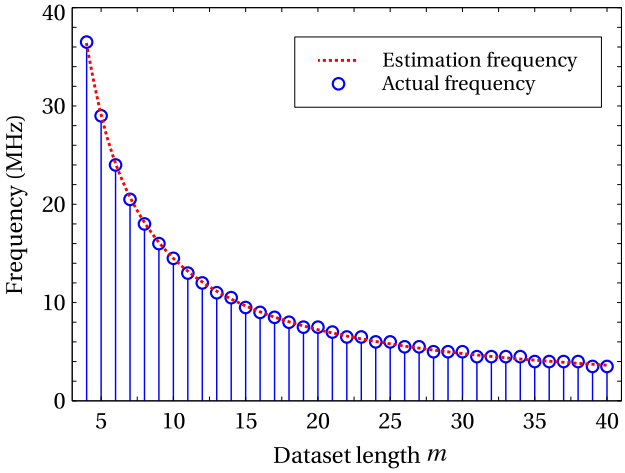

FIGURE 13. Output frequency for dataset length *m*.

TABLE 2. OpenCL kernel compilation report.

FPGA resources	Usage
logic utilization	2,434 (8%)
DSP blocks	0 (0%)
memory bits	536K (13%)
RAM blocks	86 (22%)
kernel clock	145.07 MHz

Here, f_{kernel} is the working frequency of the kernel. This working frequency can be obtained from the OpenCL kernel compilation reports as shown in Table 2. From the table, the working frequency of the kernel for this implementation is 145.07 MHz. Fig.13 shows the comparison between the frequency estimation and the actual frequency of the signal using a spectrum analyzer. The results show that the frequency estimation is similar to the actual frequency of the signal. The larger the value of m is, the lower the frequency of the output signal. Fig.14 shows examples of the output frequency for different m data lengths.

C. SIGNAL MEASUREMENT AND GENERATION

In this experiment, we demonstrate signal measurement and signal generation using OpenCL ADC and DAC components that are executed simultaneously. Kernel-to-kernel data passing using a channel extension without global memory usage is also presented.

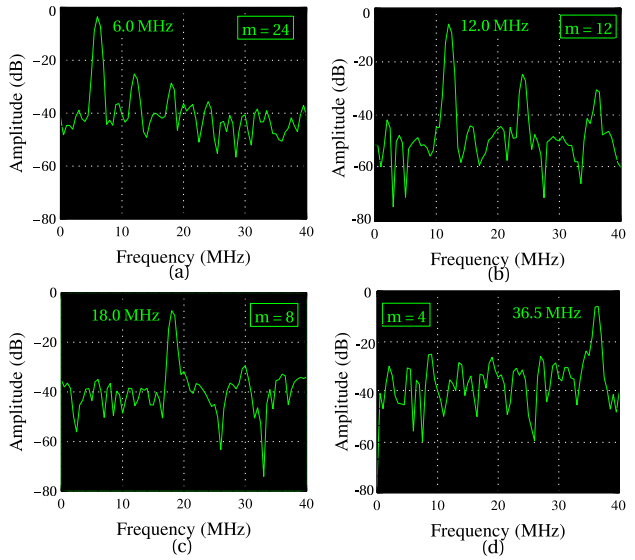

FIGURE 14. Frequency of the output signal for different data lengths: (a) $m = 24$, (b) $m = 12$, (c) $m = 8$, and (d) $m = 4$.

1) EXPERIMENTAL DESIGN

Fig.15 shows a simultaneous signal measurement and generation execution. The measured signal is passed through an I/O channel extension and is generated directly without accessing global memory. To measure and to generate a signal simultaneously, we developed three OpenCL kernels in a single OpenCL file, as shown in Listing 3. Here, we demonstrate efficient kernel-to-kernel data communication through a channel extension without accessing global memory for storing and reading data. In the *adc_channel* kernel, the signal from the ADC is read and stored in the *chan_input* channel. In the opposite direction, the *dac_channel* kernel reads the data from the *chan_output* channel and sends the data to the DAC to generate a signal.

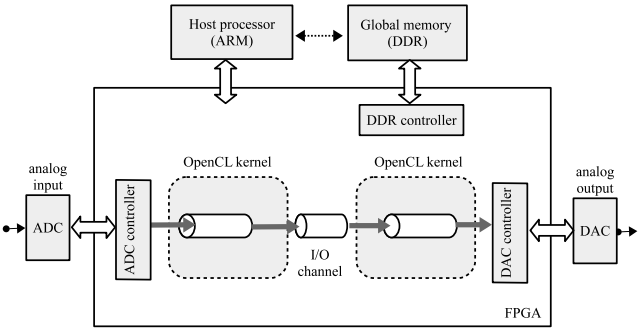

FIGURE 15. I/O channel implementation for measuring a signal, passing data, and generating a copy of the signal.

To pass data between the *adc_channel* kernel and the *dac_channel* kernel, the *in_out* kernel performs a data copy from the *chan_input* channel to the *chan_output* channel. In this implementation, the *autorun* attribute is declared for the *in_out* kernel so that the kernel is automatically executed without a host invocation.

```
// adc_dac_channel.cl
#pragma OPENCL EXTENSION cl_intel_channels : enable

channel ushort ch_data_write __attribute__((depth(0)))
__attribute__((io("ch_dac_write")));
channel ushort ch_data_read __attribute__((depth(0)))
__attribute__((io("ch_adc_read")));
channel ushort chan_input;
channel ushort chan_output;

__attribute__((max_global_work_dim(0)))
__kernel void adc_channel() {
  while(1)
  {
    write_channel_intel(chan_input,
        read_channel_intel(ch_data_read));
  }
}

__attribute__((max_global_work_dim(0)))
__attribute__((autorun))
__kernel void in_out() {
  while(1)
  {
    ushort input = read_channel_intel(chan_input);
    write_channel_intel(chan_output, input);
  }
}

__attribute__((max_global_work_dim(0)))
__kernel void dac_channel () {
  while(1)
  {
    write_channel_intel(ch_data_write,
        read_channel_intel(chan_output));
  }
}
```

Listing 3. OpenCL kernel for signal measurement and generation.

2) IMPLEMENTATION AND RESULT

To evaluate the kernel, we conducted an experiment by sending an analog input signal generated by an arbitrary signal generator through the ADC chip on the FPGA board. To measure the output signal from the DAC chip, we employed an oscilloscope and a spectrum analyzer. Fig.16 shows the comparison between the analog input signal and the analog output signal. It can be seen that the input signal is similar to the output signal. The signal measured by the *adc_channel* kernel is passed through a channel extension and generated by the *dac_channel* kernel. We also investigated the frequency

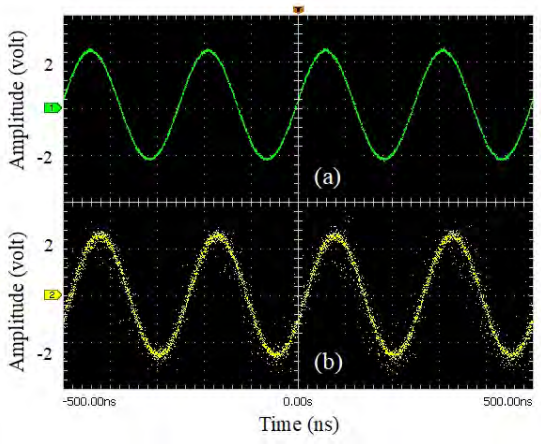

FIGURE 16. Comparison between (a) input sine wave and (b) output sine wave.

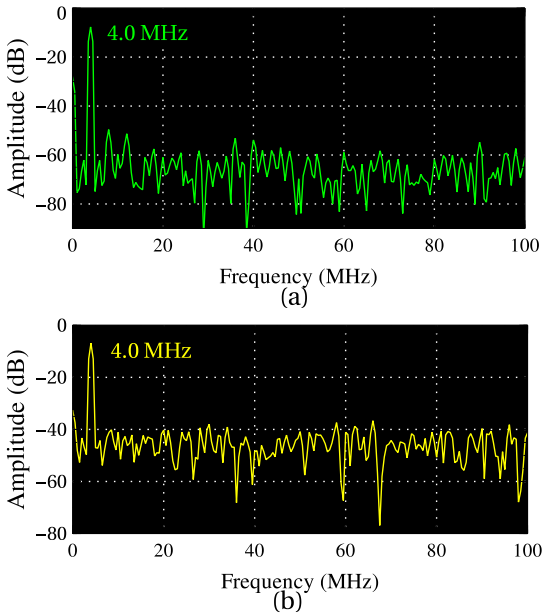

FIGURE 17. Comparison between (a) input frequency and (b) output frequency.

for both input and output signals. Fig.17 provides the frequency comparison between the input and the output signals. It can be seen that the frequencies for both signals are the same.

VI. DISCUSSIONS

In this study, we have demonstrated the implementation of an FPGA for signal generation and measurement using OpenCL programming. We developed the OpenCL components in the FPGA's BSP, which can access the FPGA's I/O for both reading and writing data. In the implementation, the OpenCL I/O channel extension is used in the kernel to read and write data from and to the components. Our implementation provides evidence that OpenCL programming can be used for generating and measuring a signal using the I/O channel extension.

We have also taken advantage of OpenCL's BSP usage because the BSP provides an external memory controller for accessing the global memory of an FPGA. BSP also provides a communication interface between the FPGA and the host that allows the host to send and receive data for analysis. Therefore, this process reduces the overall FPGA development time. This implementation focuses on design methodology, not performance accuracy. Therefore, some limitations are worth noting. According to the experiments, the working frequency of the FPGA cannot be determined at the initial condition. Thus, to achieve better data accuracy, further research should be conducted to implement an accurate clock source and FIFO buffer on the ADC/DAC board. Additionally, to avoid attenuation in signal amplitude, amplifier circuits should be employed.

VII. CONCLUSIONS

The implementation of FPGA-based signal measurement and signal generation using OpenCL is demonstrated. We performed experiments using OpenCL ADC and DAC components that can access an FPGA's I/O directly. To allow the OpenCL kernel to stream data to the FPGA's I/O, OpenCL I/O channel extensions are employed for both reading and writing data. In the signal measurement experiment, the measured signal is demonstrated to be similar to the input signal. For signal generation, the frequency of the generated signal is similar to the estimation frequency. We also demonstrate the implementation of the I/O channel extension for data transmission between two kernels. In the experiment, the measured signal is passed to a channel extension and is generated simultaneously without having to access global memory, and we verified that the frequency of the input signal is similar to the frequency of the output signal. This study has shown that the OpenCL programming language can be used for accessing an FPGA's I/O, particularly for signal measurement and signal generation. OpenCL usage reduces the development time. In this study, we focus on the OpenCL design methodology needed to access the FPGA hardware. Future research should focus on further improving performance and accuracy. Such a study can be carried out by employing a high-speed ADC/DAC board with higher sampling rate and an improved signal conditioning circuit.

REFERENCES

[1] M. C. Herbordt *et al.*, "Achieving high performance with FPGA-based computing," *J. Comput.*, vol. 40, no. 3, pp. 50–57, Mar. 2007.
[2] R. Ricart-Sanchez, P. Malagon, pp. Salva–Garcia, E. C. Perez, Q. Wang, and J. M. A. Calero, "Towards an FPGA-Accelerated programmable data path for edge-to-core communications in 5G networks," *J. Netw. Comput. App.*, vol. 124, pp. 80–93, Dec. 2018.
[3] W. Zheng, R. Liu, M. Zhang, G. Zhuang, and T. Yuan, "Design of FPGA based high-speed data acquisition and real-time data processing system on J-TEXT tokamak," *Fusion Eng. Design.*, vol. 89, no. 1, pp. 689–701, May 2014.
[4] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J. Seo, "ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler," *Integration*, vol. 62, pp. 14–23, Jun. 2018.
[5] N. Fujii and N. Koike, "IoT remote group experiments in the cyber laboratory: A FPGA-based remote laboratory in the hybrid cloud," in *Proc. Intl. Conf. Cyberworlds.*, Sep. 2017, pp. 1–6.
[6] P. S. Lee, C. K. Lee, and J. H. Lee, "Development of FPGA-based digital signal processing system for radiation spectroscopy," *Radiation Meas.*, vol. 48, pp. 12–17, Jan. 2013.
[7] A. A. Khedkar and R. H. Khade, "High speed FPGA-based data acquisition system," *Microprocessors Microsyst.*, vol. 49, pp. 87–94, Mar. 2017.
[8] V. Kandadai, M. Sridharan, S. M. Parvathy, and R. Pitchaimuthu, "A comprehensive embedded solution for data acquisition and communication using FPGA," *J. Appl. Res. Tech.*, vol. 15, no. 1, pp. 45–53, Feb. 2017.
[9] L. Qing, C. Kai, and L. Ying-Yong, "FPGA software architecture for software defined radio," *Proc. Eng.*, vol. 29, pp. 2133–2139, Feb. 2012.
[10] S. Zereen, S. Lal, M. A. S. Khalid, and S. Chowdhury, "An FPGA-based controller for a 77 GHz MEMS tri-mode automotive radar," *Microprocessors Microsyst.*, vol. 58, pp. 34–40, 2018.Apr.
[11] C. A. Ryan, B. R. Johnson, D. Riste, B. Donovan, and T. A. Ohki, "Hardware for dynamic quantum computing," *Rev. Sci. Instrum.*, vol. 88, no. 1, 2017, Art. no. 104703.
[12] M. Pelcat, C. Bourrasset, L. Maggiani, and F. Berry, "Design productivity of a high level synthesis compiler versus HDL," in *Proc. Intl. Conf. Embedded Comp. Syst. Archit., Modeling Simul. (SAMOS)*, Jul. 2016, pp. 1–6.

[13] H. M. Waidyasooriya, M. Hariyama, and K. Kasahara, "Architecture of an FPGA accelerator for molecular dynamics simulation using OpenCL," in *Proc. IEEE 5th Intl. Conf. Comput. Inf. Sci. (ICIS)*, Jun. 2016, pp. 1–5.
[14] F. Kono, N. Nakasato, K. Hayashi, A. Vazhenin, and S. Sedukhin, "Performance evaluation of tsunami simulation using OpenCL on GPU and FPGA," in *Proc. IEEE 11th Intl. Symp. Embedded Multicore/Many-Core Syst.-on-Chip.*, Sep. 2017, pp. 106–113.
[15] H. R. Zohouri, A. Podobas, and S. Matsuoka, "Combined spatial and temporal blocking for high-performance stencil computation on FPGAS using OpenCL," in *Proc. ACM SIGDA Intl. Symp. Field-Program. Gate Arrays.*, Feb. 2018, pp. 153–162.
[16] F. B. Muslim, L. Ma, M. Roozmeh, and L. Lavagno, "Efficient FPGA implementation of OpenCL high-performance computing applications via high-level synthesis," *IEEE Access*, vol. 5, pp. 2747–2762, 2017.
[17] V. Dang and K. Skadron, "Acceleration of frequent itemset mining on FPGA using SDAccel and Vivado HLS," in *Proc. IEEE 28th Intl. Conf. Appl.-Specific Syst., Archit. Processors*, Jul. 2017, pp. 195–200.
[18] K. Hill, S. Craciun, A. George, and H. Lam, "Comparative analysis of OpenCL vs. HDL with image-processing kernels on Stratix-V FPGA," in *Proc. IEEE 26th Intl. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2015, pp. 189–193.
[19] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama, "OpenCL-based FPGA-platform for stencil computation and its optimization methodology," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1390–1402, May 2017.
[20] Z. Liu, "Design of typical waveform generator based on DDS/SOPC," in *Proc. IEEE Proc. Intl. Conf. Mechtron. Sci., Electr. Eng. Comput. (MEC)*, Dec. 2013, pp. 642–646.
[21] X. Wang and Q. Mei, "High-precision design of DDS based on FPGA," in *Proc. IEEE 3rd Global Congr. Intell. Syst.*, Nov. 2012, pp. 386–389.
[22] H. F. Zhang *et al.*, "High-speed arbitrary waveform generator based on FPGA," in *Proc. IEEE Nuclear Sci. Symp. Med. Imaging Conf. (NSS/MIC)*, Oct. 2013, pp. 1–5.
[23] K. Takeda, "OPENCORE NMR: Open-source core modules for implementing an integrated FPGA-based NMR spectrometer," *J. Magn. Reson.*, vol. 192, no. 2, pp. 218–229, Jun. 2008.
[24] L. Wang *et al.*, "FPGA-based design and implementation of arterial pulse wave generator using piecewise Gaussian-cosine fitting," *Comput. Biol. Med.*, vol. 59, pp. 142–151, Apr. 2015.
[25] N. Sato, T. Chigira, K. Toyoda, Y. Iijima, and Y. Yumina, "Multi-valued signal generation and measurement for PAM-4 serial-link test," in *Proc. IEEE 48th Intl. Symp. Multiple-Valued Logic (ISMVL)*, May 2018, pp. 1–10.
[26] T. Lee *et al.*, "EPICS data acquisition system for FPGA-based plasma electron density measurement on KSTAR," *Fusion Eng. Design.*, vol. 129, pp. 320–325, Apr. 2018.
[27] R. Akeela and B. Dezfouli, "Software-defined radios: Architecture, state-of-the-art, and challenges," *Comput. Commun.*, vol. 128, pp. 106–125, Sep. 2018.
[28] S. Sridharan, "Evaluation of OpenCL for FPGA for data acquisition and acceleration in high energy physics," in *Proc. 21st Intl. Conf. Comput. High Energy Nuclear Phys.*, May 2015, pp. 1–5.
[29] (2018). *Intel FPGA SDK for OpenCL, Programming Guide*. [Online]. Available: https://www.intel.com
[30] (2018). *Intel FPGA SDK for OpenCL Pro Edition Custom Platform Toolkit User Guide*. [Online]. Available: https://www.intel.com
[31] S. O. Settle, "High-performance dynamic programming on FPGAs with OpenCL," in *Proc. High Perform. Extr. Comp. Conf (HPEC)*, May 2013, pp. 604–615.
[32] H. M. Waidyasooriya, M. Hariyama, and K. Uchiyama, *Design of FPGA-Based Computing Systems With OpenCL*. New York, NY, USA: Springer, 2018, pp. 89–90.
[33] R. Kobayashi, Y. Oobata, N. Fujita, Y. Yamaguchi, and T. Boku, "OpenCL-ready high speed FPGA network for reconfigurable high performance computing," in *Proc. Intl. Conf. High Perform. Comput. Asia-Pacific Region.*, 2018, pp. 45–56.
[34] (2018). *Terasic, DE1-SoC OpenCL User Manual*. [Online]. Available: http://www.terasic.com.tw/en/
[35] (2018). *Terasic, THDB-ADA User Manual*. [Online]. Available: http://www.terasic.com.tw/en/
[36] S. Leither, H. Wang, and S. Tragoudas, "Design techniques for direct digital synthesis circuits with improved frequency accuracy over wide frequency ranges," *J. Circuits, Syst. Comput.*, vol. 26, no. 2, Feb. 2017, Art. no. 1750035.
[37] B. Cronin. (2018). *DDS Devices Generate HighQuality Waveforms Simply, Efficiently, and Flexibly*. [Online]. Available: http://www.analog.com
[38] (2018). *Fundamentals of Direct Digital Synthesis (DDS)*. [Online]. Available: http://www.analog.com

IMAN FIRMANSYAH received the M.Sc. degree from the Department of Physics, University of Indonesia, in 2008, and the M.E. degree from the Department of Information Science, Chiba University, in 2012. He is currently pursuing the Ph.D. degree in computer science with the University of Tsukuba. Since 2005, he has been with the Indonesian Institute of Sciences (LIPI). His research interests include embedded systems, signal processing, and heterogeneous computing using GPU and FPGA.

YOSHIKI YAMAGUCHI received the M.S. degree in science and engineering and the Ph.D. degree in engineering from the University of Tsukuba, in 2000 and 2003, respectively. From 2000 to 2003, he was a JSPS Research Fellow with the University of Tsukuba. From 2003 to 2005, he was a Research Scientist with the Yokohama Institute, RIKEN. Since 2005, he has been an Assistant Professor with the Graduate School of System and Information Engineering, University of Tsukuba. He was a Visiting Academic with the Department of Computing, Imperial College London, from 2010 to 2011. Since 2011, he has been a Collaborative Fellow with the Center for Computational Science, University of Tsukuba. His research interests include reconfigurable architecture, real-time applications, and power-efficiency computing for image, sound, and bioinformatics applications. He is also interested in heterogeneous computing, including FPGA, GPU, and CPUs.