

2021 | WHITE PAPER

FPGA-centric software acceleration made easy

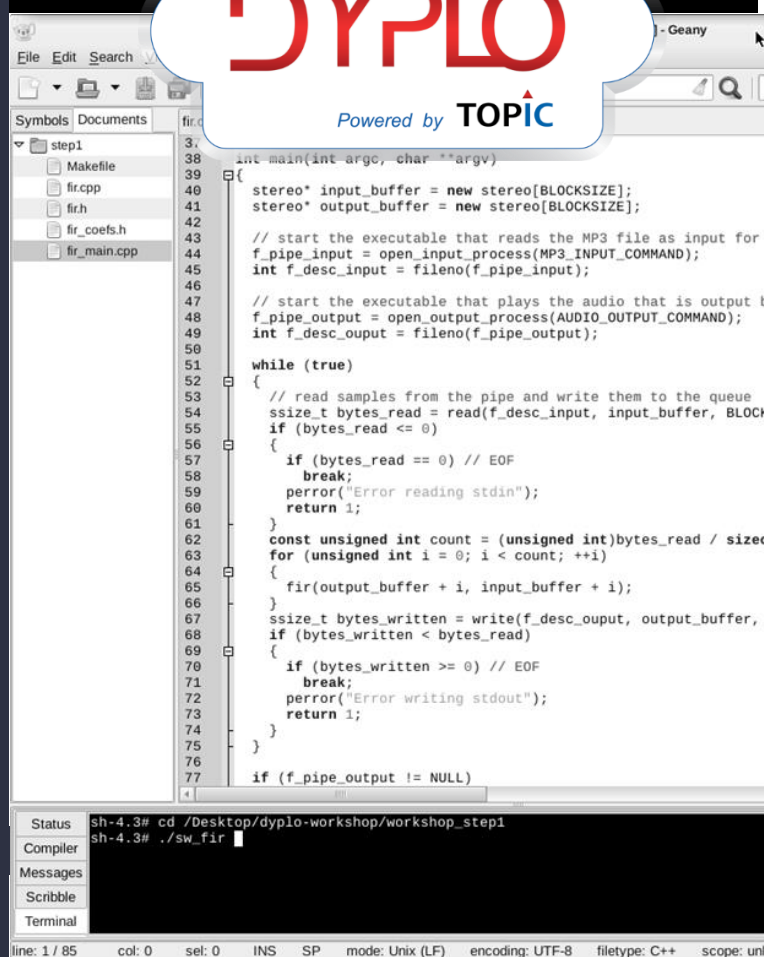
Build (embedded) high-performance computing software solutions using FPGA acceleration technology with standard software programming methods

Dirk van den Heuvel | Product manager


TOPIC

DYPLO[®]

Powered by **TOPIC**



```
int main(int argc, char **argv)
{
    stereo* input_buffer = new stereo[BLOCKSIZE];
    stereo* output_buffer = new stereo[BLOCKSIZE];

    // start the executable that reads the MP3 file as input for
    f_pipe_input = open_input_process(MP3_INPUT_COMMAND);
    int f_desc_input = fileno(f_pipe_input);

    // start the executable that plays the audio that is output t
    f_pipe_output = open_output_process(AUDIO_OUTPUT_COMMAND);
    int f_desc_output = fileno(f_pipe_output);

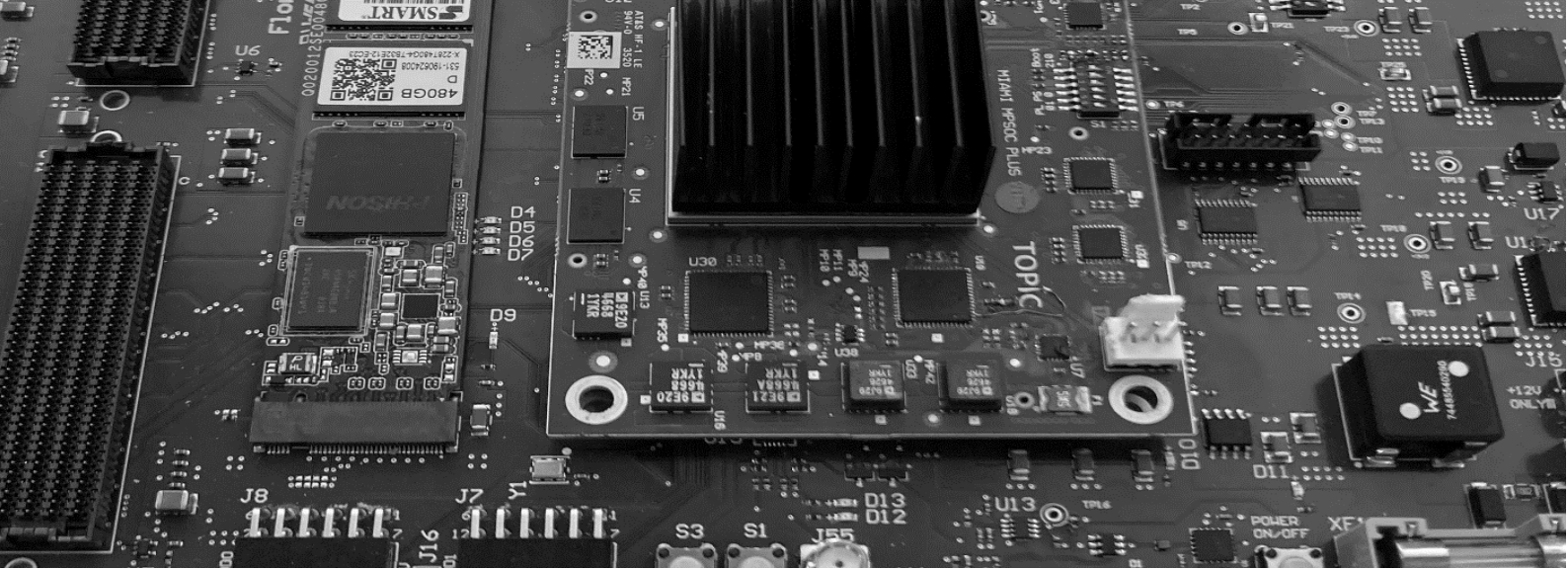
    while (true)
    {
        // read samples from the pipe and write them to the queue
        ssize_t bytes_read = read(f_desc_input, input_buffer, BLOCKSIZE);
        if (bytes_read <= 0)
        {
            if (bytes_read == 0) // EOF
                break;
            perror("Error reading stdin");
            return 1;
        }
        const unsigned int count = (unsigned int)bytes_read / sizeof(stereo);
        for (unsigned int i = 0; i < count; ++i)
        {
            fir(output_buffer + i, input_buffer + i);
        }
        ssize_t bytes_written = write(f_desc_output, output_buffer, count * sizeof(stereo));
        if (bytes_written < bytes_read)
        {
            if (bytes_written >= 0) // EOF
                break;
            perror("Error writing stdout");
            return 1;
        }
    }

    if (f_pipe_output != NULL)
```

CONTENTS

CONTENTS	2
1. Introduction.....	3
2. Bridging the CPU-FPGA integration gap	3
3. Dynamic Process Loader (Dyplo®).....	4
3.1 Dyplo® Network-on-Chip.....	5
3.2 Linux driver	5
3.3 Function integrator.....	6
4. Programming example	7
5. Dyplo® in action.....	11
About TOPIC Embedded Systems.....	12

For more information go www.TOPIC.nl or contact us via
+31 (0)499 33 69 79 | info@TOPIC.nl



1. Introduction

The world of computing is becoming more and more heterogeneous. Processing platforms integrate with an increasing pace varieties of processing units integrated into single System On Chips (SOCs), (embedded) PCs and combinations of edge and cloud computing. Within this context, you see a shift in programming methods, a broad range of software abstractions and a focus on coding efficiency like “low-coding”. An interesting question to this development is how to program the different processing architectures with a common and effective software approach? For multi-core processor architectures, a wide variety of compiler solutions are already available. However, when you look at heterogeneous accelerator platforms, which combine GPUs, FPGAs and neural networks with multi-core CPUs, the programming method needs specific programming skills.

This white paper addresses the programming of FPGA fabric in a software development perspective and explains a method to develop and integrate FPGA functionality in a typical software development context without requiring significant FPGA experience.

2. Bridging the CPU-FPGA integration gap

The main difference between a CPU and FPGA is that a CPU executes instructions on a predefined logic silicon structure, whereas on an FPGA the logic structures also need to be designed. However, everything executed on the FPGA is running in parallel, thereby boosting performance significantly. This also means that the programming abstraction level of the FPGA is lower compared to a CPU. The use of FPGA devices is therefore mostly focused on high-speed signal processing, video applications, communication interfaces, blockchain algorithms and other compute-intensive applications with regular constructs. To program FPGA devices, the use of a specific programming language like VHDL and Verilog are required. In addition, the use of IP libraries and OpenCL type of kernels can simplify the effort required - yet low-level FPGA design know-how remains crucial.

Over the years, significant energy has been put into creating compilers for FPGA implementations using C or C++ as the preferred programming language, for example Xilinx High Level Synthesis (HLS). Here a CPU compiler builds functionality on top of an existing fixed processor architecture, and an FPGA compiler must also compile the logic structure. Currently the technology is mature and while the logic densities of FPGA devices are very high, a bit of logic inefficiency for such technologies is acceptable.



3. Dynamic Process Loader (Dyplo®)

The challenge that remains is that after the compilation of the FPGA functionality, this resulting functionality still needs to be integrated with the processing system. On both a System-on-Chip (SOC) as well as on a PC/FPGA combination, data needs to be exchanged between the two processing entities. Typically specialist expertise is required at this stage: writing of Linux kernel drivers, construction of proper DMA based data exchange mechanisms, additions of high-performance FPGA interfaces according to strict bus protocols, etc. – effectively here is where multiple programming disciplines meet. TOPIC recognized this problem years ago and developed a Dynamic Process LOader (Dyplo®) that solves this problem elegantly on the FPGA side, Dyplo® forms a Network-on-Chip (NOC), wrapping fixed and dynamically exchangeable FPGA function blocks. On the processor side, Dyplo® is a Linux kernel driver that interfaces with the Dyplo® NOC using file I/O based data streams. The third aspect of Dyplo® is the implementation flow to transform a software defined function block into a Dyplo® wrapped FPGA function block.

In this paper, the focus will be on the integration of FPGA logic with a PC system using PCI-Express as the interface medium. However, the design process and functionality are identical to SOC type of implementations or in a cloud context. The examples mentioned are implemented using a standard Intel i7 based PC system running Ubuntu 18.04LTS and incorporating a Xilinx Alveo U50 FPGA accelerator card with a 16 lanes PCI-Express 3.0 bus.

The Dyplo® concept is based on streaming data transport. The FPGA communication infrastructure is loosely based on Kahn Processing Networks (KPN). This means that nodes (accelerator regions) are interacting via buffers, which synchronize operation between nodes and match computational performance of the system with the available communication bandwidth. In the software application, the data to and from the FPGA are accessible as file streams. You need to open them, read from or write to, and then close them. The streams are presented as standard Linux file streams with a clear reference.

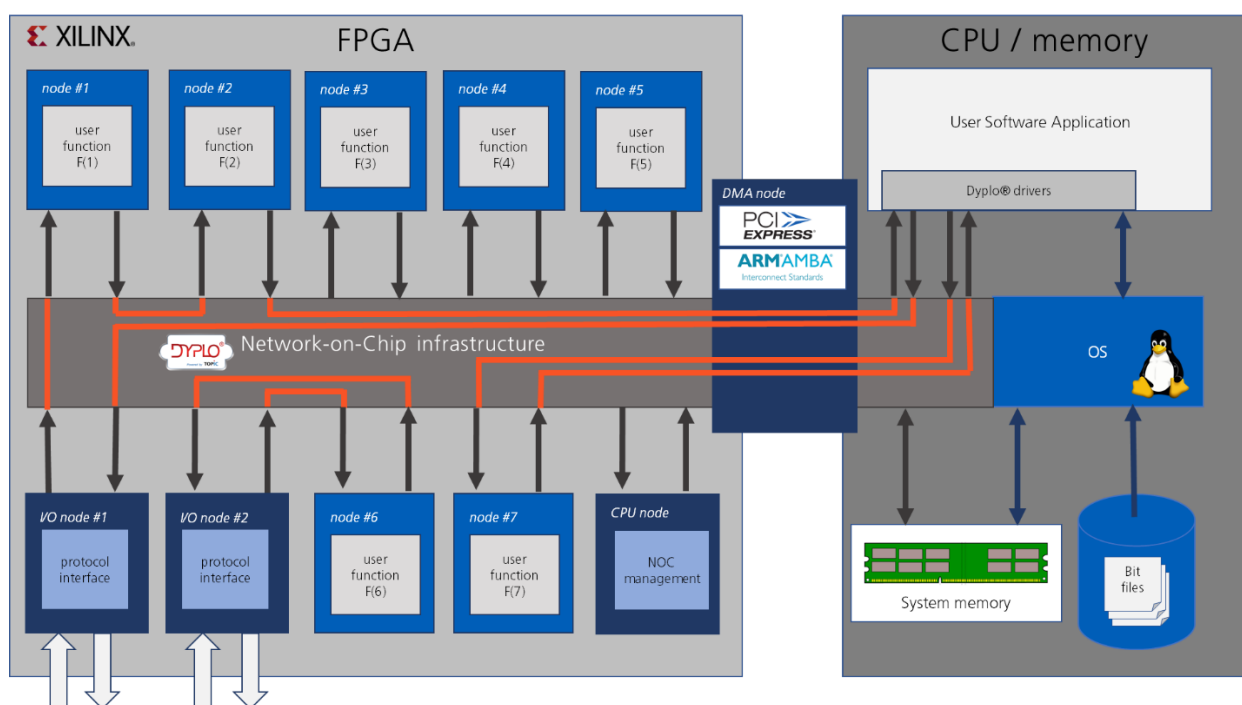


Figure 1 – Dyplo® conceptual architecture

3.1 Dyplo® Network-on-Chip

Figure 1 gives a conceptual architecture illustration of the Dyplo® NOC from a system perspective. Using single to multiple parallel DMA data streams, data can flow between the PC and FPGA fabric. The processors volatile system background memory is used for the data exchange. The Dyplo® infrastructure directly interacts with this background memory with limited processor involvement. Of note is that using Linux, the observed responsiveness and latency is much better then when using Microsoft Windows as an operating system.

The NOC is constructed as a ring topology with a configurable bandwidth. The default data width of the ring is 64 bits. In multiples of 64 bits, the ring performance can be increased. The clock rate of the ring is dependent on the FPGA fabric technology and the device speed grade. Clock rates of over 300MHz are possible. Each node in the NOC is connected to this ring with maximum of 4 input and 4 output streams. There are five types of nodes that interact with this communication infrastructure.

Node type	Description
DMA nodes	Reserved for data exchange between CPU and FPGA fabric
I/O nodes	Intended for direct data exchange between Dyplo® nodes and non-Dyplo® functionality on the FPGA.
Fixed nodes	User-defined/programmed functionality that resides permanently in the NOC. This functionality is always available.
Reconfigurable nodes	User-defined/programmed functionality that can be dynamically exchanged in the NOC with different functions. This node has no function, unless the software application programs the node with a partial bitstream
CPU node	Reserved node for managing the NOC behavior and synchronization with the software system

Routing the inputs and outputs of nodes with other nodes or the CPU system is controlled by the application software. Setting-up a route is a simple addressing mechanism, stating from which node (1 to 32) and which output (1 to 4) to which node (1 to 32) and which input the data should flow. These routes are flexible and can be changed on-the-fly by software commands.

3.2 Linux driver

The Linux driver for Dyplo® creates a software abstraction of the NOC interfaces and loading of the reconfigurable nodes. The Linux driver allows multiple applications to share the same Dyplo® infrastructure. The arbitration of modifying routes and node functionality is controlled by Dyplo®. If an application tries to allocate a specific already occupied node, the driver will return an error flag. The same is valid for programmed routes.

The Dyplo® driver builds on top of standard Linux drivers for memory and DMA control as well as PCI Express drivers. For proper handling of the parallel data streams and specifics of Dyplo® an additional driver layer is created around these drivers. Using these drivers, the DMA data exchange channels are available for the user as file streams.



3.3 Function integrator

A third part of Dyplo® is the tool, the Dyplo® Development Environment (DDE), to create and manage the partial bitstreams of the FPGA. A GUI is available for interactive configuration, compilation and bitstream management. In addition, scripted and command line driven operation is supported. The tool helps to configure the NOC FPGA IP in a comprehensive manner, guides the import and creation of function blocks from C/C++ and HDL and it creates automatically the corresponding partial bitstreams for the compiled baseline FPGA image. Figure 2 gives an impression of the DDE user interface.

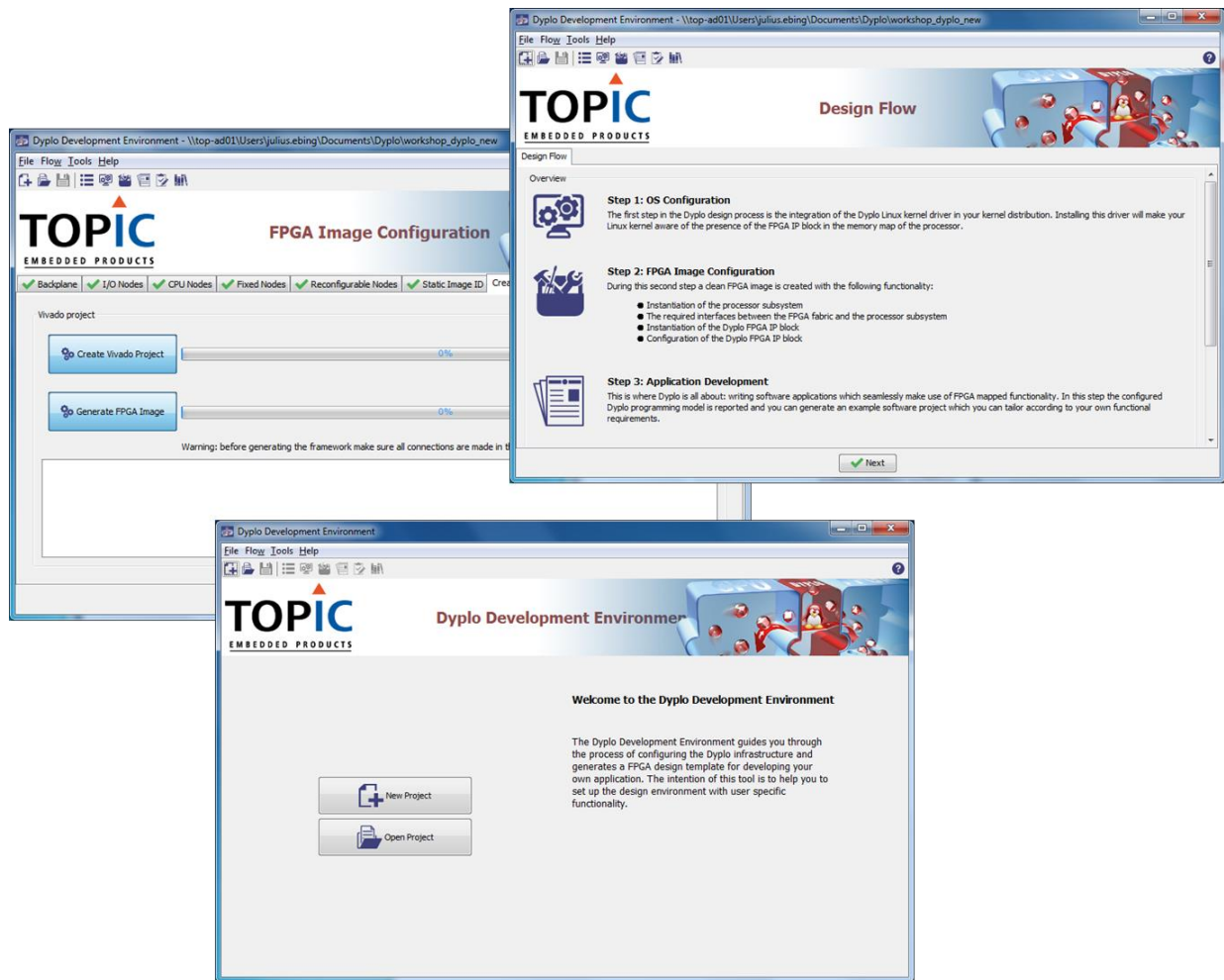


Figure 2 – Example screenshots of the Dyplo® Development Environment

4. Programming example

The question remaining is, how does an actual design flow workout in practice? In the following paragraphs, a description is given of a typical design flow using the Dyplo® acceleration framework based on a desktop PC solution with an Alveo U50 board. However, the exact same design flow is applicable if you want to execute this in a cloud configuration at e.g. Nimbix or Amazon, or if you are working on a system with a Zynq 7000 device or a Zynq Ultrascale+ device.

4.1 Installation

This particular Dyplo® workflow is explained using the Dyplo® Development Kit. The kit consists of an Alveo U50 board, the Dyplo® Development Environment (DDE) and an illustrative example application. In addition, a PC running Linux (e.g. Ubuntu 18.04 LTS) with an available 8 lanes PCI-Express 3.0 slot is required. Make sure that GCC is installed on the machine in addition to a Xilinx Vivado/Vitis installation, preferably version 2020.2 or later. Install the Dyplo® Linux driver by simply running the Dyplo® installer which is part of the DDE. The required Dyplo® specific PCI-Express driver is automatically installed and you will notice that a number of file I/O devices are created as standard Linux peripherals.

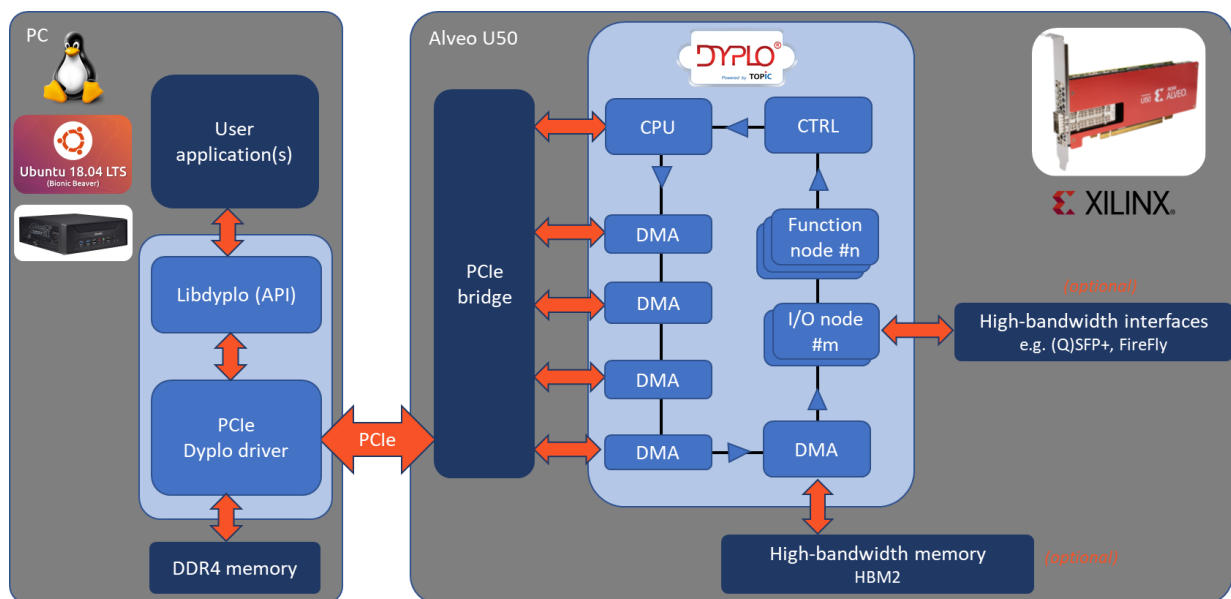


Figure 3 – Demo image configuration Dyplo® Development Kit

4.2 NOC configuration

The second step is to configure the Dyplo® NOC on the FPGA, according to the needs of your application. You basically partition the large FPGA into smaller partitions with well defined, easy to use streaming interfaces. This configuration process is guided using an interactive GUI, launching scripts that result in a complete FPGA project and bit image for the Alveo U50 board. The development kit also comes with a pre-configured NOC wrapping 8 reconfigurable nodes and 4

DMA nodes. Figure 3 illustrates the functionality of this reference configuration. This means you can have 4 parallel streams to the FPGA fabric as well as 4 back into the PC. The full PCI-Express bandwidth is available, providing more than 500Mbyte/sec bandwidth per channel when equally divided over the 4 DMA channels, matching performance requirements of 4Kp60 video applications.

4.3 Function implementation

This is where the user specific functionality comes in. Dyplo® supports two implementation flows. The first implementation flow is based on the traditional FPGA design flow, but strongly simplified. The interfaces of the HDL node shall be AXI4-Stream compatible. A template design with test bench are provided. The code can be hand-crafted, can be created using functions from Xilinx standard IP catalog, can be constructed using Simulink HDL coder or bought from a third party.

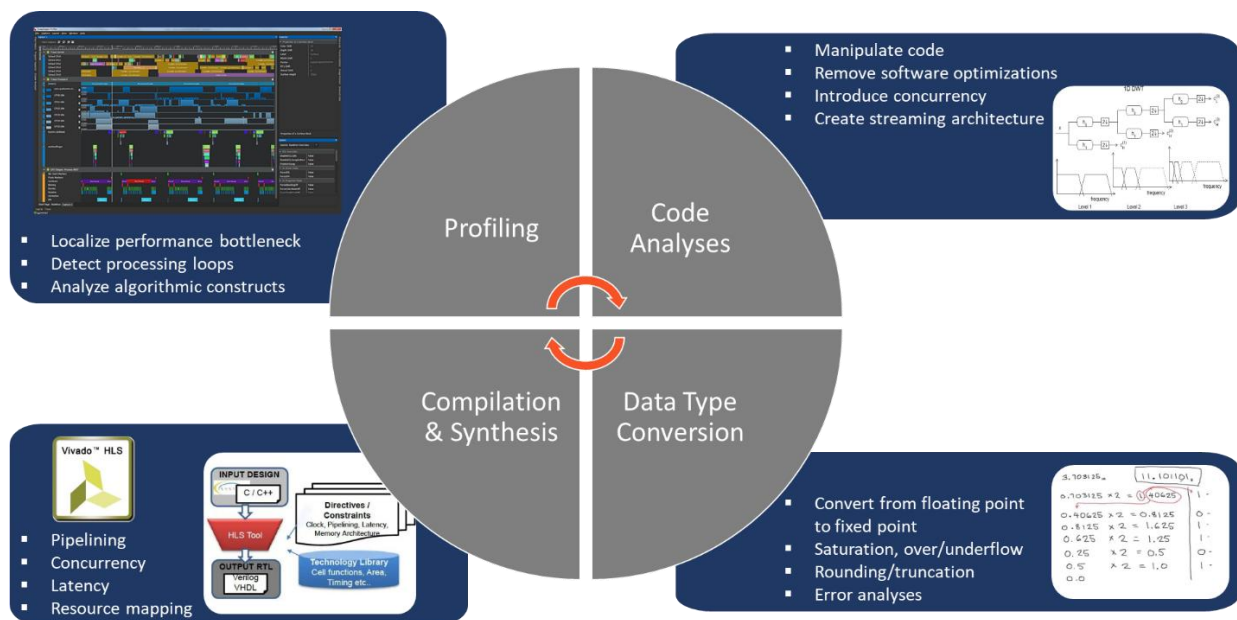


Figure 4 – Typical iterative implementation cycle of FPGA accelerator functions

The second implementation flow uses Xilinx high level synthesis technology (HLS). In the application code, the part of which the performance needs to be accelerated, has to be isolated and interfaced according to the Dyplo® API requirements. This is very similar to the way file I/O functionality is handled in Linux. Using the Dyplo® GUI the provided code is automatically wrapped for compliance with the NOC interfaces. In the same pass, a Vivado HLS project is created for this particular function with the default settings. By opening the Vivado HLS GUI, the function can be further optimized to meet specific processing performance requirements. All features offered by Vivado HLS are supported, including a selection of OpenCL and OpenCV operators. A typical design cycle to get from a “software” function to an FPGA accelerated equivalent is illustrated in figure 4.

Both implementation flows result in partial bit streams for the particular function that can be deployed on any of the 8 reconfigurable nodes. The reference design comes with a number of

standard video manipulation functions, provided as C/C++ code, HDL code and as partial bitstreams.

4.4 Application development

In the previous step, a function was isolated from the application software for acceleration. To replace the software function by the Dyplo® accelerated FPGA variant, the application needs to meet the Dyplo® programming model requirements. It is very similar to file I/O operations:

- Start the driver. By default, the NOC on the FPGA is in a passive state. It needs to be initialized using a specific command.
- Configure the routing of streams within the NOC:
 - o From the software application to one of the nodes (maximum 4 streams)
 - o From the nodes back to the software application (maximum 4 streams)
 - o Between the nodes in the NOC on the FPGA (each node has maximum 4 input streams and 4 output streams)

The result of this operation are file-type pointers that can be used by software in the applications

- The file pointers can be activated using standard file operators as “open” and “close”. When opened, you can read and write these streams at will. Operations on these streams are blocking in such a way that data stops flowing when a consuming data process is not able to keep-up. This prevents data loss and synchronization issues. If dropping of data is required, this must be implemented explicitly by the programmer. Implicitly, Dyplo® is loss-less. You can keep on pushing data to the nodes in the FPGA, until the driver blocks.
- The final operation to get the application using the accelerator in the fabric is the actual programming of the node with a partial bitstream. This is done by a simple programming command, referencing the node number and the partial bitstream file. Successful programming is flagged by the return value. A node can already be occupied by a different application as Dyplo® allows multiple applications to use the same NOC.

Although the programming model is inspired on file I/O, the combination with the NOC configuration has similarities with a specific OpenCL programming construct. Therefore, the use of FPGA devices in a software context using Dyplo® is of a comparable complexity as developing CUDA or OpenCL applications for GPU accelerator boards.



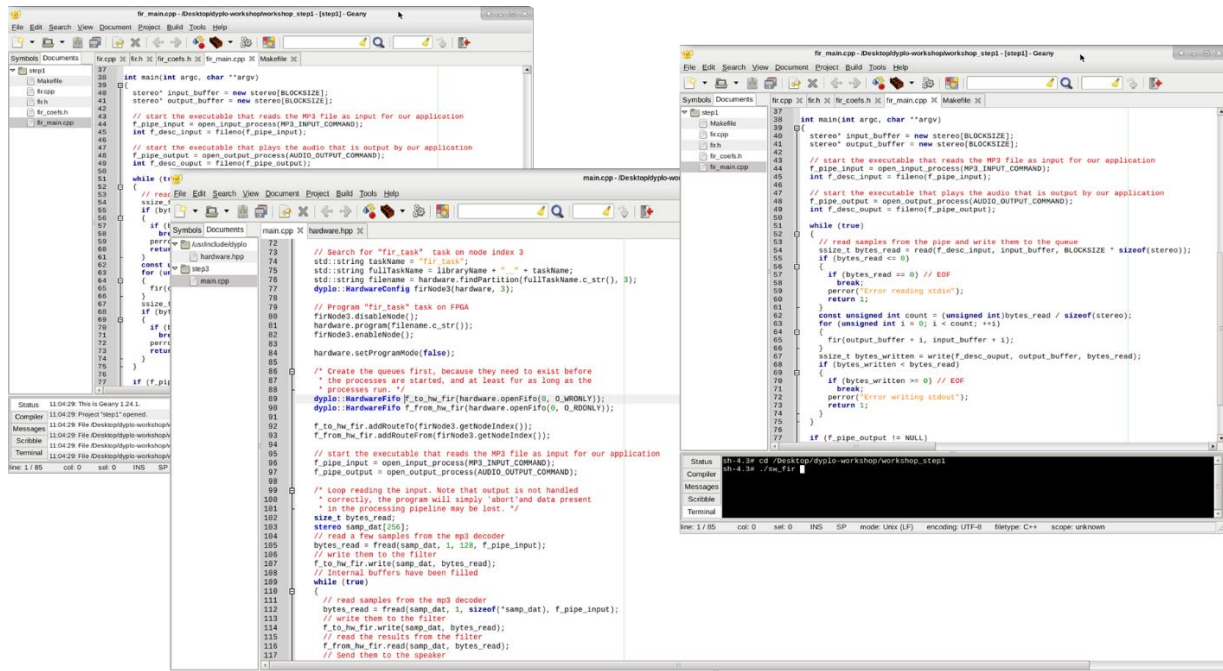


Figure 5 – Common software IDE tools are used to create Dyplo® compatible application software

5. Dyplo® in action

In the previous chapters, the Dyplo® concept and programming method is explained. However, the best way to learn about the concept is to try it. Dyplo® is available for Zynq 7000, Zynq Ultrascale+ and Alveo. Via the Xilinx App Store and the licensing using Accelize technology, the Dyplo Development and Runtime Environment can be sourced. However, the best way to experience the Dyplo® benefits is using the Dyplo® Development Kit.



Figure 6 – A Qt-based example application running 2 applications simultaneously in the NOC

TOPIC released recently Dyplo® 2.0, with improved NOC performance, extended devices support for Alveo accelerator boards and 4K video support capabilities. Where the 1.x version of Dyplo® focused on the disclosure of FPGA fabric for software development, the 2.0 version of Dyplo® is unlocking the performance and integration capabilities of the FPGA fabric and seamlessly integrating this within Dyplo®. Able to maintain the threading type of processing infrastructure based dynamic function exchange (DFX, formerly known as partial reconfiguration), the data communication infrastructure is further enhanced to support natively multi-4K video streams, make the DMA-based software-in-the-loop via DDR memory an integral part of the system, and allows seamless integration of high-bandwidth-memory (HBM) in the data path as well as connecting multiple FPGA devices to each other using high bandwidth, error-free connection links. As a Dyplo® 2.0 introduction offer, the Dyplo® Development Kit is offered including a Xilinx Alveo U50 board.

Unleash superior accelerated algorithmic performance on FPGAs now by testing Dyplo® yourself. Use either a SOC or PC based system and experience with GPU programming convenience and reduced power consumption the benefits of such a flow.



About TOPIC Embedded Systems

“We make the world a little better, healthier and smarter every day”. Our mission statement reflects exactly what we do: developing innovative systems for our customers. The way we do that, is by combining our customers domain specific know-how with our expertise in hardware and software development. This results in the most optimal product for our customers. TOPIC has a strong background of more than 25 years in developing systems, which can contain embedded-, application- and cloud software, FPGA code and PCB designs. We help customers in different domains such as medical, imaging, machine control & safety.

With over 150 employees, we are a strong and established company with our headquarters in Best, the Netherlands. TOPIC has an ISO13485 (medical) certified Quality Management System and adopted the Agile way-of-working for optimal interaction with the customer.

Premier Alliance Partnership with Xilinx | TOPIC is one of the few Xilinx Premier Alliance Partners in the world and the only one in the Benelux. Our partnership with Xilinx started in 2008 and since then we have been working closely together for decades.



Materiaalweg 4, 5681 RJ Best, the Netherlands



+31 (0)499 336979



info@TOPIC.nl



www.TOPIC.nl



linkedin.com/company/TOPIC-embedded-systems