DOI: 10.35598/mcfpga.2026.009

Another FPGA Development Environment For Edge Artificial Intelligence

Volodimir Karnaushenko
ORCID 0000-0001-7744-2569
Department of Microelectronics,
electronic devices and appliances
Kharkiv National University of Radioelectronics
Kharkiv, Ukraine
vladimir.karnaushenko@nure.ua

Ihor Shcherban
ORCID 0000-0002-6896-3095
Department of Microelectronics,
electronic devices and appliances
Kharkiv National University of Radioelectronics
Kharkiv, Ukraine
ihor.shcherban@nure.ua

Ihor Bondarenko
ORCID 0000-0003-3907-6785
Department of Microelectronics,
electronic devices and appliances
Kharkiv National University of Radioelectronics
Kharkiv, Ukraine
ihor.bondarenko@nure.ua

Oksana Babychenko
ORCID 0000-0002-4806-6549
Department of Microelectronics,
electronic devices and appliances
Kharkiv National University of Radioelectronics
Kharkiv, Ukraine
oksana.babychenko@nure.ua

***Abstract*— Edge computing is a platform where data storage, processing, and analysis are moved closer to the data source, rather than relying on centralized cloud servers. This move helps reduce latency, conserve bandwidth, and improve real-time application responsiveness. Edge computing enables the Internet of Things (IoT) and other connected devices to process data locally and respond in real-time, reducing latency, increasing reliability, and minimizing bandwidth usage. In today's interconnected world, this localized approach has become essential for efficient and scalable data management.**

Keywords — AI, FPGA, RISC-V, LVDS, MIPI, GPIO, JTAG, VHDL, RTL, IoT.

I. Introduction

Autonomous and robotic systems require high-throughput, low-latency sensor data for spatial control and navigation. Real-time decisions are made using input from a variety of sensory modalities, including cameras, radar, lidar, microphones, and ultrasonic sensors. Much of this data is transmitted from distributed edge systems to centralized computing platforms over high-speed serial connections that also provide bidirectional control and error signaling. As perceptual systems become more complex, the bandwidth, number, and complexity of these serial channels are rapidly increasing. With this growth comes a wider range of potential failure modes [1].

II. Challenges in edge AI

The demand for artificial intelligence (AI) at the edge is growing exponentially, driven by applications such as autonomous vehicles, industrial automation, and Internet of Things devices. As edge devices demand faster processing, lower latency, and power efficiency, field-programmable gate arrays (FPGAs) have become a compelling choice for accelerating AI workloads. With their reconfigurable hardware and parallel processing capabilities, FPGAs can deliver customized performance that combines speed, power, and flexibility. When robotics and autonomous systems require high-performance artificial intelligence, FPGAs can act as the primary data processing interface. FPGAs aggregate and transmit high-speed sensor data to accelerators to enable, for example, real-time robotic vision and multisensory data fusion.

Edge computing is a platform where data storage, processing and analysis are moved closer to the data source, instead of relying on centralized cloud servers. This step helps reduce idle time, save bandwidth, and improve the responsiveness of real-time applications.

Edge computing enables the Internet of Things (IoT) and other connected devices to process data locally and respond in real-time, reducing latency, increasing reliability, and minimizing bandwidth usage. In today's interconnected world, this localized approach has become essential for efficient and scalable data management [2].

Efinix has released a family of FPGAs focused on peripheral AI applications based on software or hardware RISC-V cores with AI instruction extensions (Fig. 1).

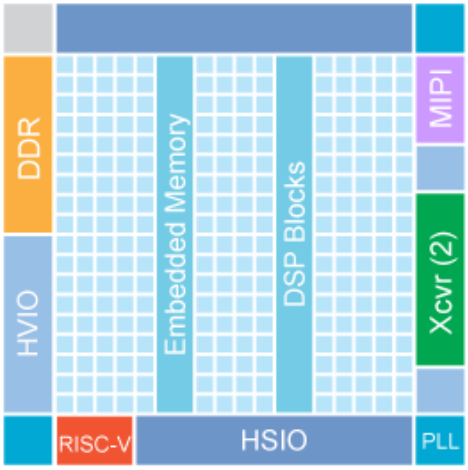

Fig. 1. Package Planner

In addition to hardware, the company provides developers with an integrated design environment with a full set of relevant features (Fig. 2).

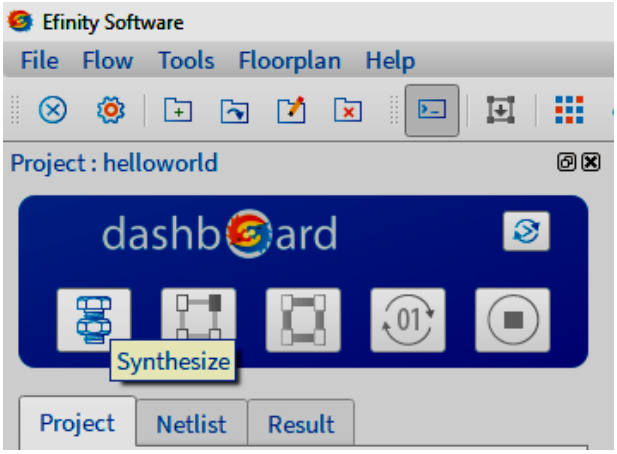

Fig. 2. Dashboard Controls

Efinity® software provides a complete set of tools for designing using FPGAs and Efinix® cores. The graphical user interface (GUI) provides a visual way to configure projects, run the programming process, view floor plan information, and create interfaces surrounding the logic of the project [3]. Efinity® software version 2026.1 has the following new features and improvements:

• Improved execution time and memory usage;

• Support for new devices;

• Unified netlist flow supports LVDS;

• Unified modeling supports all simple peripherals: GPIO, HVIO, HSIO, PLL, clock multiplexer, LVDS, MIPI lane, oscillator, and JTAG;

• Synthesis support for marking control points;

• Improved support for installation in a shared environment;

• Programmer with fast firmware verification mode.

To work with Efinity, you need to have the appropriate hardware and software.

General requirements:

• Full release of Efinity: 64-bit operating system, at least dual-core;

• Text editor such as Notepad, gVim, Visual Studio;

• Computer memory requirements (when compiling Efinity projects): 8 - 12 GB.

Windows OS requirements:

• Windows 10 or later, 64-bit operating system;

• Microsoft Visual C++ 2022 x64 runtime library;

• Zadig software for installing USB drivers;

• 64-bit Java runtime, required for configuring some IP cores in IP Manager (e.g. Sapphire SoC).

Before creating a project, you need to set general tool settings to control the software workflow.

The Efinity main window contains controls on the toolbar to start the tool workflow, including synthesis, placement, routing, and bitstream generation.

Project management features:

• Project Editor;

• Project Tab;

• Linking to RTL Source Files;

• Using VHDL Libraries;

• Packaging Design Files;

• Migration Project to Other FPGA Tools.

Place and Route Tab (Optional). The options on this tab allow you to specify project-specific settings to help close the time frame. If no settings are made, the tool will use default values (Fig. 3).

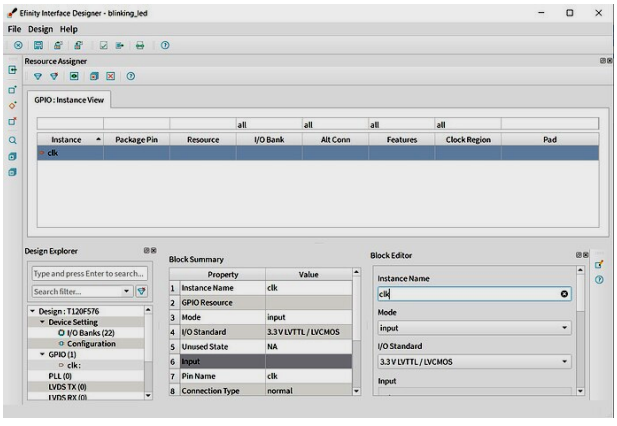

Fig. 3. GPIO instance created

The Efinity® software includes documentation as PDF user guides and on-line HTML help. This documentation is provided with the software (Fig. 4).

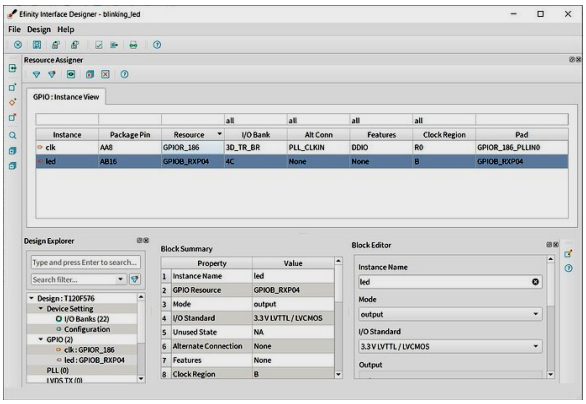

Fig. 4. Pin assignments

III. Some Efinity Tools

How to Use the Debugger [4].

The Efinity software includes a hardware debugger that simplifies probing signals in FPGA designs via the JTAG interface. The debugger includes two debug cores:

• Configuring virtual I/O (vio) cores using a manual process and a profile editor;

• Configuring logic analyzer cores using a manual or automated process in the debug wizard.

Automated Debugger

To create a debug profile:

1. Open a project.

2. Synthesize the design.

3. Launch the debug wizard by clicking its icon in the main icon bar.

4. Select "Designed Netlist" or "Postcard" in the "Signals From" list.

5. Select the *LED* and counter buses from the list on the left and move them to the right using the >> button.

6. Leave the default probe type, *DATA AND TRIGGER*.

7. Proceed to the next step. The wizard generates a debug profile.

8. Make sure that "Automatically create instances" is enabled to integrate the debug profile into your project. Click "Finish".

9. The program asks to recompile. Click "OK".

Programming the FPGA for debugging

1. Start debugging by selecting *Tools* > *Open Debug*.

2. Make sure that the board is recognized as a USB target. If not, connect the board and click "Refresh USB Targets".

3. Click the "Select Image File" button.

4. Browse to the source stream directory and select the bitstream file.

5. Click "Start Programming". The console will display programming messages.

TABLE1. FPGA CONFIGURATION MODES

Mode	Description
SPI Active (serial/parallel)	The FPGA loads the bitstream itself from non-volatile SPI flash memory.
SPI Passive (serial/parallel)	An external microprocessor or microcontroller sends the bitstream to the FPGA using the SPI interface.
JTAG	A host computer sends instructions through a download cable to the FPGA's JTAG interface using JTAG instructions.

Observing probed signals in debug

1. Initiate a debug connection.

2. In the Trigger Settings tab, add the desired nets.

3. Specify the trigger conditions (e.g. values).

4. Start debugging to collect data during startup.

5. When finished, *GTKWave* will automatically open to display the waveform (Fig. 6).

6. Disconnect the debugger to stop working.

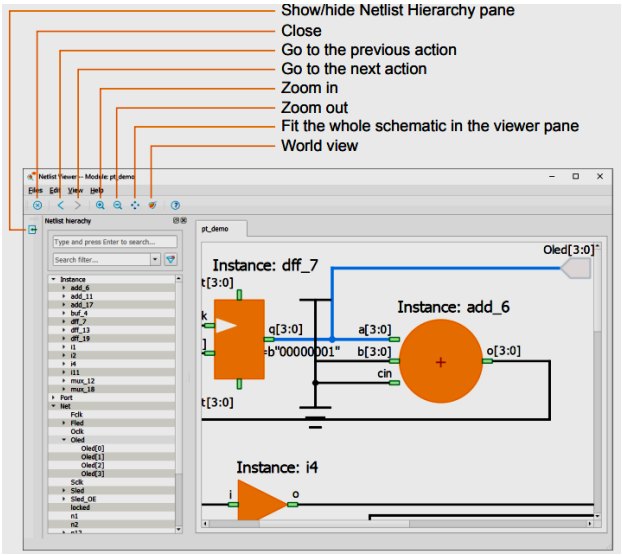

Fig. 6. The Netlist Viewer tool displays and analyzes your design's

netlist

Manual Debugger

Creating a Debugger Profile

To configure the Virtual I/O and Logic Analyzer debug kernels in a profile:

1. Open the project.

2. Start the debugger via *Tools* > *Open Debugger*. Since no debug profile exists, the Profile Editor window opens (Fig. 5).

3. Add a Virtual I/O (VIO) kernel using Add Debug Kernel > Virtual I/O. Configure the probes (input) and sources (output) signals as specified (i.e. name and width).

4. Add a Logic Analyzer (*LA*) core via *Add Debug Core* > *Logic Analyzer*. Configure the probes to capture signals that match the VIO settings.

5. Generate the debug RTL to create the necessary debug files.

6. Open debug_*top.v* and rename "*edb_top*" to "*edb_top_manual*".

7. Close the debugger.

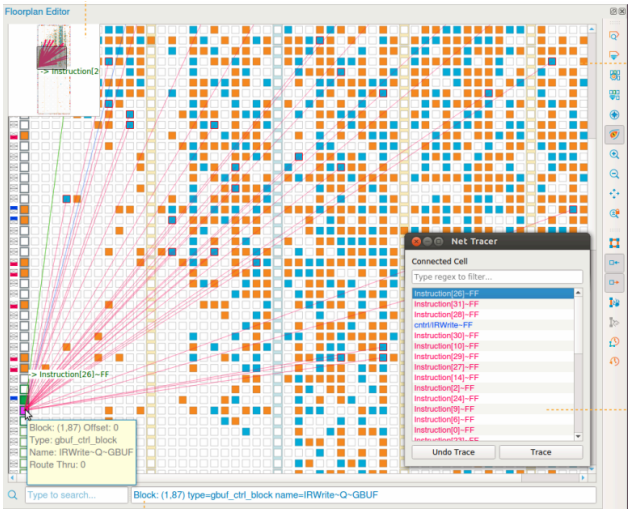

Fig. 5. Floorplan Editor

Adding the Debug Core to a Project

To integrate the debug code into your project and compile it:

1. Open the Efinity main window and go to the Project tab.

2. In the Interface Designer, add a JTAG User Tap block, after configure it, and generate the SDC constraints (Fig. 7).

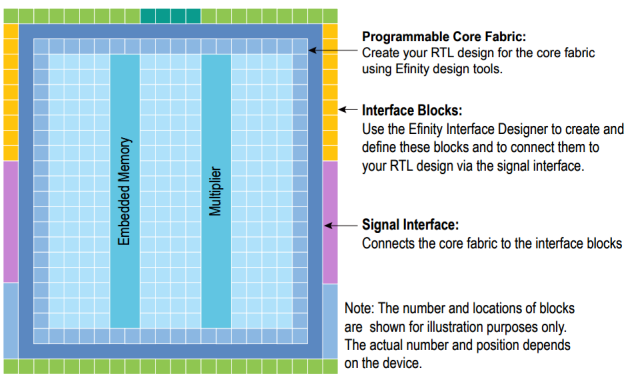

Fig. 7. Interface Designer

3. Close the Interface Designer.

4. Modify the project file by uncommenting certain lines to enable debugging code and create an instance of the debug_top module.

5. Save the changes.

6. Compile the project.

Program the FPGA

To program the FPGA:

1. Open the debugger by selecting *Tools* > *Open Debugger*.

2. Verify that the Trion T20 development board is recognized as a USB target; refresh the page as needed. To assign an FPGA pin to a port, click in the Resource column of the GPIO Instance View that corresponds to the port and type the name of the FPGA pin. The Trion T120F576 Development Kit has a 50 MHz oscillator on pin GPIOR_186. Assign this to the *clk* instance. Assign pin GPIOB_RXP04 (LED0 on the kit) to the *led* instance (Fig. 8) [5].

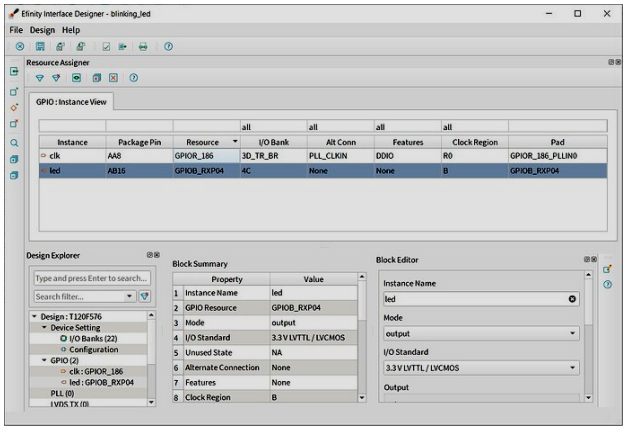

Fig. 8. Pin assignments

3. Select the FPGA configuration file (bitstream) from the output stream directory using the "Choose Image File" button (Fig. 9).

4. Start programming by clicking "*Start Programming*" and observe the programming messages in the console.

Fig. 9. Debugger Profile Editor Perspectice

IV. Starting the Debugger

To observe probed signals in the debugger:

1. Connect the debugger and go to the la0 tab to configure the trigger (Fig. 10).

2. Add the desired trigger condition.

3. If you want to change the value of vio0, start the debugger.

4. Then wait for the trigger to fire, after the trigger is applied, open *GTKWave* to check the debug signal.

5. Disconnect the debugger to stop recording data.

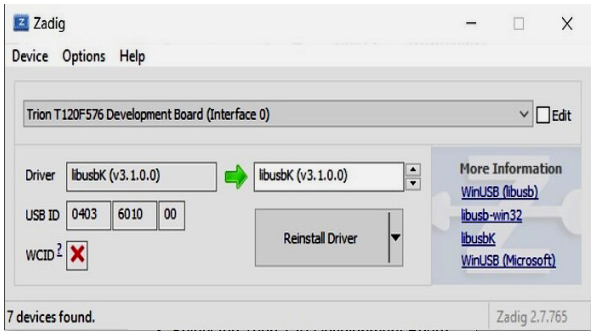

Fig. 10. Zadig software for libusbK driver installation

Efinix provides a free development environment for its FPGA line, but customer need to purchase a development board to use it. The cheapest board is called the Fireant ($25USD). It's a T8 FPGA, a USB programmer, 2 custom switches, 4 custom LEDs, and some 3.3V I/O on a 40-pin board that's suitable for connecting to a prototype board. It doesn't require any external components to run, and it can be powered via a USB programming cable. If you need a larger board for your demo, there are other development boards with the Trion T8, T20, and T120 FPGAs [6].

Conclusion

FPGAs are transforming the deployment of AI at the edge, offering unparalleled flexibility, low latency, and power efficiency. As AI applications at the edge continue to expand, the ability to leverage FPGA accelerators will be a critical skill for embedded engineers. While challenges remain, advances in FPGA technology and development tools are making their potential easier and more accessible.

Having yet another embedded developer tool for use in AI edge devices is a definite boon to the engineering community as it moves toward the intelligent, connected systems of the future.

References

[1] https://www.eetimes.com/ai-on-the-edge-is-iot-ready/
[2] https://www.design-reuse.com/article/61270-fpga-comes-back-into-its-own-as-edge-computing-and-ai-catch-fire/
[3] https://www.efinixinc.com/products-riscv.html
[4] https://www.efinixinc.com/products-riscv.html#pills-riscv
[5] https://www.efinixinc.com/docs-html/efinity/topics/hlp-xcvr-dbg-intro.html
[6] https://ve7it.cowlug.org/efinix.html