White Paper

AI and Embedded
Nios® V and Arm*

Deploying tinyML* on Altera® FPGAs

Enabling AI on Altera® FPGAs with Soft and Hard Embedded Processors

Authors

Liang Yu Gooi
Software Enabling and Optimization Engineer
Altera Corporation

Fawaz Jumaah
Software Enabling and Optimization Engineer
Altera Corporation

Shreya Mehrotra
Software Product Manager
Altera Corporation

Introduction

In an era of fast technological innovation, artificial intelligence (AI) is becoming the transformative driver that can reshape our society and nearly every industry. AI is already the main driver of emerging technologies such as image recognition, robotics, and IoT, and generative AI has further expanded the possibilities and popularity of AI. AI has shown extraordinary abilities to train in vast amounts of data, generate new content, transform content, and make useful decisions. While most processing currently happens in the cloud, the rise of edge AI is already underway, as we see AI deployed across industries such as industrial automation, healthcare, autonomous vehicles, and retail. With security, low-latency real-time processing, low power, low cost, and small form factors all critical, alternatives to large-scale GPU deployment are needed at the edge.

Tiny machine learning (tinyML*) is a new, exciting, and growing field that helps bring machine learning to low-power devices such as microcontrollers. It can be used in various sectors, including agriculture, industrial predictive maintenance, anomaly detection, smart homes, and personalized healthcare. It can also be used for computer vision, voice and speech recognition, gesture recognition, diagnostic assistance, and more.

With their inherent architectures, FPGAs naturally implement customized, optimal AI and machine learning (ML) workloads. With integrated Arm processors, Altera's RISC-V solution, the Nios® V soft processors, thousands of digital signal processing (DSP) blocks, a variety of memory hierarchies, and support for multiple I/O standards, they are very well adapted to different AI/ML architectures, providing optimal performance and power efficiencies.

Altera's broad product portfolio is used in embedded, edge, network, and data center applications. Altera offers the FPGAs AI suite application software to help deploy medium and large deep learning AI models seamlessly on Altera® FPGAs and SoC FPGAs. Small models such as tinyML can also be deployed on power- and cost-optimized Altera® FPGAs and SoC FPGAs.

Integrating tinyML with low-power microcontrollers such as Nios V and Arm processors on Altera FPGAs – by leveraging their flexibility and performance – enables tinyML to operate on a broad spectrum of devices, bringing intelligent compute to the edge. Users can execute complex AI tasks on smaller, less resource-intensive FPGA devices, deploying a cost-effective and power-efficient solution. Users can also implement multiple Nios V processor instances running tinyML on Agilex™ FPGAs. No-cost tools and a unified debugging environment make developing and debugging solutions easy across all Altera SoC FPGAs. Additionally, by porting 32-bit tinyML applications onto the 64-bit Arm* Cortex-A architecture, Altera enables the seamless enhancement of computational efficiency and future-proofing embedded ML solutions without any changes to the software.

This white paper demonstrates deploying a tinyML model on Altera's RISC-V solution, the Nios V/g general-purpose processor, and the Arm Hard Processor System on the Agilex 5 SoC FPGA. It also discusses the versatility of FPGAs in AI deployment, the deployment steps, and the future scope of work.

Table of Contents

Introduction 1
Altera SoC FPGAs Offer Versatile AI Deployment 2
Deploying tinyML on Altera FPGAs and SoC FPGAs 2
TFLite Micro with Nios V Processors 3
TFLite Micro with the Arm HPS Processor 6
Further Enhancements 8
Conclusion 9
References 9

Altera SoC FPGAs Offer Versatile AI Deployment

Versatile AI deployment refers to the ability to deploy AI/ML models across various platforms, environments, and applications with flexibility and efficiency. Altera SoC FPGAs naturally enable AI, support diverse use cases, and support small models such as tinyML to medium and large models such as convolutional neural networks (CNN), recurrent neural network (RNNs), and multilayer perceptrons (MLPs) at the edge and network, and generative AI large language model (LLM) transformer models in the data center.

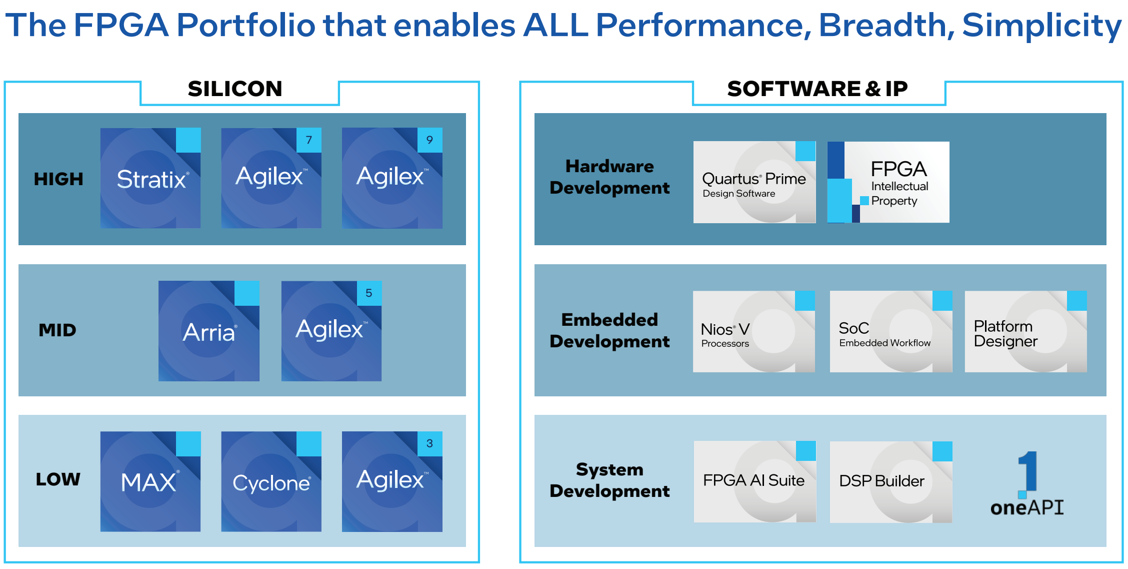
Altera delivers a broad portfolio of custom logic solutions such as FPGAs, SOCs, and CPLDs together with software tools, intellectual property (IP), and embedded processors, offering versatile and efficient solutions for accelerating AI tasks, particularly in applications where customization, deterministic, low latency, tight integration, and energy efficiency are critical. Their long product life cycles and the re-programmability help future-proof designs.

Altera's midrange FPGAs, such as Agilex 5 and Arria® 10, are optimized for applications requiring high performance, lower power, and smaller form factors. At the same time, Agilex 3, MAX®, and Cyclone® FPGAs are ideal for applications that need power- and cost-optimized devices in compact form factors.

Altera SoC FPGAs integrate the FPGA fabric with hardened Arm processors, offering a high degree of reconfigurability and performance in a single device. The Nios V processor is the next generation of soft processors for Altera FPGAs, implemented in the FPGA fabric and based on the open-source RISC-V instruction set architecture (ISA).

Altera FPGAs and SoC FPGAs enable ML developers to deploy tinyML models by applying either the Nios V soft-core processor in the FPGA fabric or the Arm Cortex-A processor in the Hard Processor System (HPS). Users can rapidly innovate by deploying the models on RISC-V or Arm processors. Once the model becomes stable, in the case of soft processors, custom instructions can be used to deploy the model to further accelerate its performance by redeploying the slower code portion of the model onto the FPGA fabric as a hardware accelerator.

The Ashling RiscFree* Integrated Development Environment (IDE) for Intel® FPGAs is developed for Altera FPGAs. This IDE provides software development and heterogeneous debug support for the soft Nios V processors based on the RISC-V ISA and the hard Arm processors.



Deploying tinyML on Altera FPGAs and SoC FPGAs

TensorFlow* is a free and open-source library that makes developing and deploying ML models easier. TensorFlow Lite (TFLite)[1] is a subset of the TensorFlow repository intended to run on mobile and embedded devices with a larger external RAM and can run full-fledged operating systems such as Android*, iOS*, and embedded Linux*. TensorFlow Lite for Microcontrollers (TFLite Micro) is a subset of TFLite for running ML models on microcontrollers and other devices with limited internal ram. It can run bare-metal systems or real-time operating systems (RTOS). The TFLite micro library helps to run inference on TFLite models in microcontrollers.

Here are the high-level steps used for deploying a model on a Nios V or Arm processor:

- **Train model**: Train a model using a dataset in the TensorFlow framework or a pre-trained model.
- **Convert model**: Convert the TensorFlow model to the TFLite model format using the TFLite converter.
- **Check accuracy**: Check the TFLite model accuracy and minimize error due to conversion.

- **Create FPGA hardware system**: Generate a processor system with the Nios V or Arm processor in the Quartus® Prime design software and program the SoC FPGA.
- **Integrate TFLite Micro into Nios V or Arm processors**: Integrate the TFLite Micro library into the processor's software environment and the TFLite model.
- **Deploy and validate system**: Upload the integrated model into the SoC FPGA and test the model on the deployed system.
- **Perform model inferencing**: Use the model to make predictions and classifications.

A brief example of platform integration can be demonstrated using the sine wave regression model (a Hello World example prepared by TFLite). It includes the basics of using the library and a full end-to-end workflow of training a model and converting it with TFLite Micro for running inference on a microcontroller. Like a typical "hello world" C/C++ program, both Nios V and Arm processor applications require a single UART device to display messages on the console. Thus, it is suitable to verify platform integration attempts with minimal device dependency.

The TFLite Hello World example is a pre-trained model with a single fully connected layer replicating a sine function. It is also converted into the TFLite model format with minimal accuracy loss. With the **Train model, Convert model**, and **Check accuracy** steps completed, you can begin platform integration from the **Create FPGA hardware system** and onwards.

TFLite Micro with Nios V Processors

The general guidelines for using the Nios V soft-core processor in Altera FPGAs, as illustrated in Figure 1, involve:

1. Create an FPGA hardware system
   A. Choose a soft-core processor
   B. Instantiate the processor
   C. Add peripherals
   D. Generate the FPGA bitstream
   E. Program the FPGA
2. Integrate TFLite Micro into the Nios V processor
   A. Develop the tinyML software
   B. Apply the TFLite model
3. System deployment and validation
   A. Download the tinyML software into the FPGA
   B. Test the system

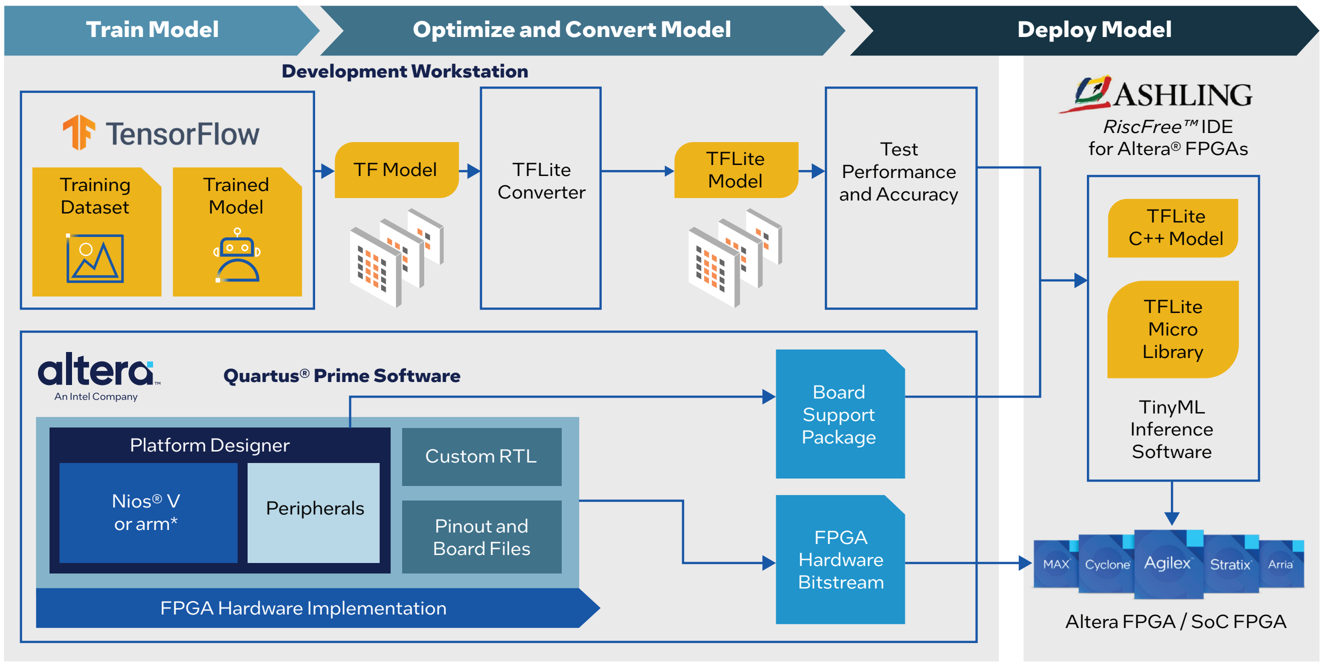


Figure 1. Training and Deploying tinyML on Nios V or Arm Processors on Altera SoC FPGAs

1. **Installation of board-specific IDE**: Install the Ashling RiscFree IDE for Intel FPGAs on your machine. To complete this task, refer to Installing Ashling RiscFree IDE for Intel FPGAs in the references section.
2. **Toolchain setup, compiler flags, and linker setup**: In the Nios V processor Board Support Package (BSP) editor, define:
   - cflags_user flags: -ffunction-sections -fdata-sections -fno-rtti -fno-exceptions
   - cflags_defined_symbols: -DTF_LITE_STATIC_MEMORY
   - cxx_flags: -std=c++17
   - link_flags: -Wl,--gc-sections
   - cflags_optimization: -O3
3. **Peripheral integration**: The BSP editor automatically selects the JTAG UART device as the standard UART device during BSP generation. You can apply the same setup.
4. **Build TFLite static library:** The following git commands should be run in a Linux or WSL environment.

```
$ sudo apt update
$ sudo apt install make git python3 unzip python3-pip
$ pip3 install numpy image
$ git clone --depth 1 https://github.com/tensorflow/tflite-micro.git
$ cd <clone directory>/tflite-micro
$ python3 tensorflow/lite/micro/tools/\
project_generation/create_tflm_tree.py \
-e hello_world tflite_app
```

This creates a parent folder named tflite_app with the following child folders shown in Figure 2.

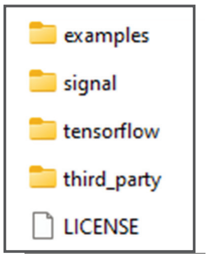

examples
signal
tensorflow
third_party
LICENSE

Figure 2. Generated TFLite Static Library

5. **Customize timing function**: Replace the timing function in micro_time.cc as shown.

```
#include <time.h>
#include "system.h"
...
uint32_t ticks_per_second() { return ALT_CPU_TICKS_PER_SEC; }
...
uint32_t GetCurrentTimeTicks() { return clock(); }
```

With the above steps, a newly generated TFLite Micro library is integrated into the Nios V processor.

6. **Build the TFLite Hello World example**
   Ensure the BSP project (named tflite_bsp) is generated successfully after implementing steps 2 and 3. Next, we can apply the generated tflite_app as the APP project and create the application CMakeLists file using the niosv-app executables in the Nios V processor command shell.

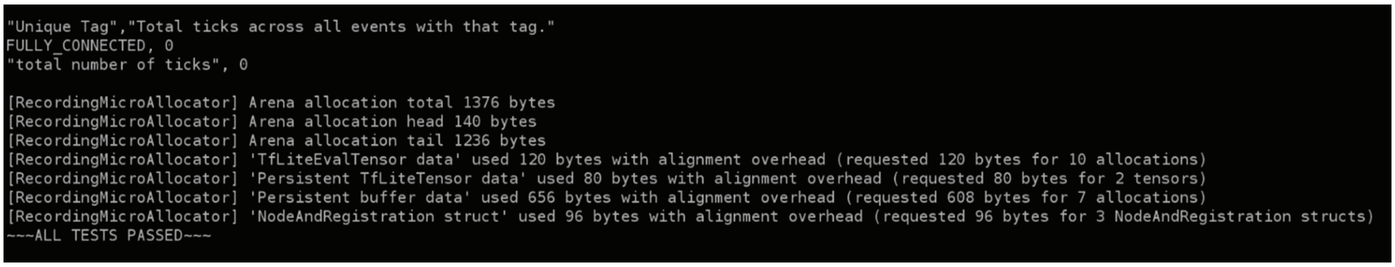
```
$ niosv-app \
--bsp-dir=tflite_bsp \
--app-dir=tflite_app \
--srcs-recursive=tflite_app/examples,\
tflite_app/signal,\
tflite_app/tensorflow \
--incs=tflite_app,\
tflite_app/examples/hello_world,\
tflite_app/tensorflow,\
tflite_app/third_party/flatbuffers/include,\
tflite_app/third_party/gemmlowp,\
tflite_app/third_party/kissfft,\
tflite_app/third_party/ruy
```

   You may import the BSP and APP projects into the Ashling RiscFree IDE for Intel FPGAs or continue building them from the command line.

```
$ cmake -S tflite_app \
-B tflite_app/build/Release \
-G "Unix Makefiles" \
-DCMAKE_BUILD_TYPE=Release
$ make -C tflite_app/build/Release
```

System Deployment and Validation

Once the TFLite Hello World example elf file is generated, you may download it into the processor core and test the example. The Nios V processor platform integration is completed successfully when the application prints "ALL TESTS PASSED", shown in Figure 3.



```
"Unique Tag","Total ticks across all events with that tag."
FULLY_CONNECTED, 0
"total number of ticks", 0

[RecordingMicroAllocator] Arena allocation total 1376 bytes
[RecordingMicroAllocator] Arena allocation head 140 bytes
[RecordingMicroAllocator] Arena allocation tail 1236 bytes
[RecordingMicroAllocator] 'TfLiteEvalTensor data' used 120 bytes with alignment overhead (requested 120 bytes for 10 allocations)
[RecordingMicroAllocator] 'Persistent TfLiteTensor data' used 80 bytes with alignment overhead (requested 80 bytes for 2 tensors)
[RecordingMicroAllocator] 'Persistent buffer data' used 656 bytes with alignment overhead (requested 608 bytes for 7 allocations)
[RecordingMicroAllocator] 'NodeAndRegistration struct' used 96 bytes with alignment overhead (requested 96 bytes for 3 NodeAndRegistration structs)
~~~ALL TESTS PASSED~~~
```

Figure 3. TFLite Hello for the Nios V Processor

TFLite Micro with the Arm HPS Processor

The hardware design used for this experiment is the Altera SoC FPGA golden hardware reference design (GHRD). The tinyML application runs under the Zephyr* OS environment as a single-thread application. The only dependency used from Zephyr is the clock ticks application program interface (API) to measure the execution time.

The complete flow of the HPS development is portrayed in Figure D and can be summarized below:

1. Create an SoC FPGA hardware system
   A. Choose the Altera SoC FPGA GHRD
   B. Generate the SoC FPGA bitstream
   C. Prepare the Arm Trusted Firmware (ATF) bootloader
   D. Integrate the SoC FPGA bitstream with the ATF first-stage boot loader (FSBL) application
   E. Program the SoC FPGA

2. Integrate TFLite Micro into the Arm processor
   A. Prepare the Zephyr OS
   B. Modify the Zephyr OS
   C. Develop the tinyML software
   D. Apply the TFLite model

3. System deployment and validation
   A. Download the ATF second-stage boot loader (SSBL) application into the SoC FPGA
   B. Download the tinyML software into the FPGA
   C. Test the system

Altera's Agilex 5 SoC FPGA is chosen as the example SoC FPGA for the following guideline.

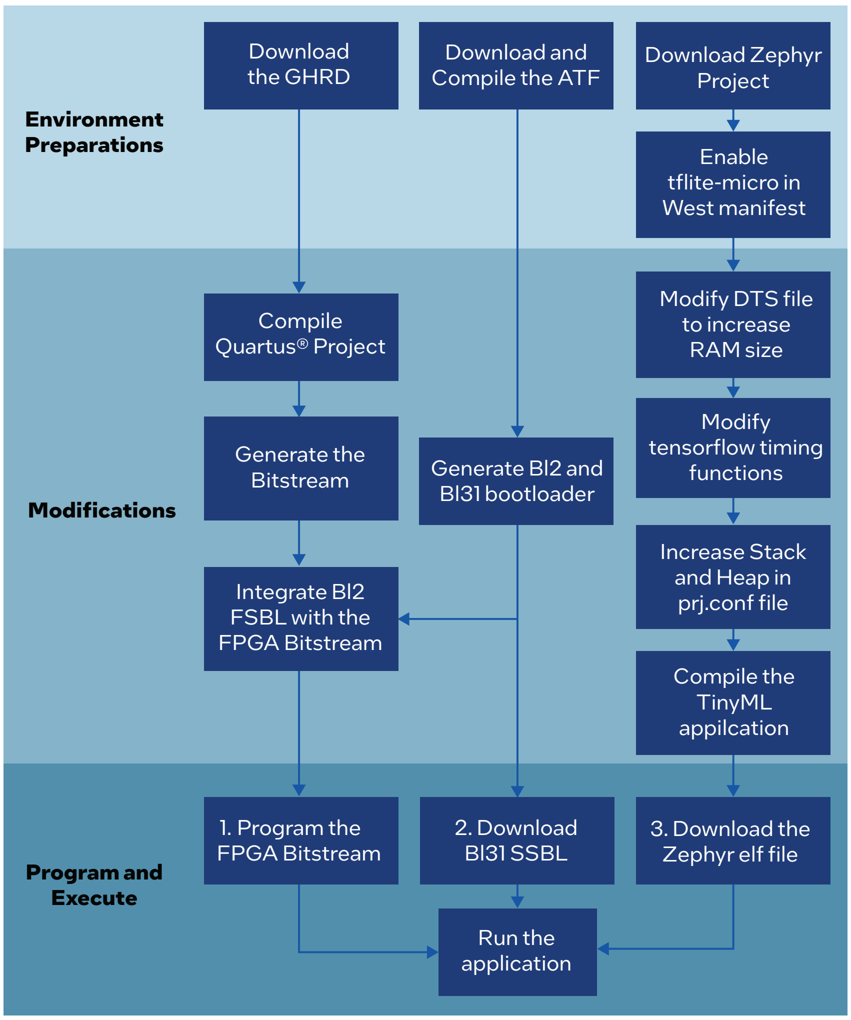

Environment Preparations
Download the GHRD
Download and Compile the ATF
Download Zephyr Project
Enable tflite-micro in West manifest
Modifications
Compile Quartus® Project
Generate the Bitstream
Integrate Bl2 FSBL with the FPGA Bitstream
Generate Bl2 and Bl31 bootloader
Modify DTS file to increase RAM size
Modify tensorflow timing functions
Increase Stack and Heap in prj.conf file
Compile the TinyML appilcation
Program and Execute
1. Program the FPGA Bitstream
2. Download Bl31 SSBL
3. Download the Zephyr elf file
Run the application

Figure 4. Development Flowchart for the Arm HPS

Create an SoC FPGA Hardware System

To begin platform integration for HPS processors, download the Altera Agilex 5 SoC FPGA GHRD from the Golden Hardware Reference Design (GHRD) Build Scripts Github repository in the references section and compile the Quartus Prime design software project to generate the SoC FPGA bitstream. Concurrently, the HPS requires the Arm Trusted Firmware to generate the HPS bootloader: BL2 as an FSBL and BL31 as an SSBL. BL2 must be embedded with the SoC FPGA bitstream to initialize and configure the HPS once booted.

Integrate TFlite Micro into the Arm Processor

A Zephyr project is needed to compile a Zephyr application. Once installed, modifications are needed to enable TFLite libraries within the Zephyr OS. The steps to perform a TFLite platform integration for Zephyr and the Agilex 5 FPGA HPS are as follows:

1. **Installation of board-specific IDE**: Install the Zephyr Software Development Kit (SDK) and its dependencies on your machine. To complete this task, refer to Developing with Zephyr – Getting Started Guide in the references section.

2. **Device tree modifications**: By default, the Agilex 5 device tree RAM size is 8 MBytes, which is insufficient for a common tinyML application. Increase the RAM size in the following device tree file.

```
<install_path>/zephyrproject/\
zephyr/dts/arm64/intel/\
intel_socfpga_agilex5.dtsi
```

3. **Stack/heap size modifications**: Increase the stack/heap size in the Zephyr project configuration files (prj.conf).

```
CONFIG_MAIN_STACK_SIZE=65536
CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE\
=524288
CONFIG_HEAP_MEM_POOL_SIZE=262144
```

4. **Peripheral integration**: The device tree file selects the UART controller as the standard UART device by default. You can apply the same setup.

5. **Toolchain setup, compiler flags, and linker setup**: This build uses the default settings of a Zephyr project. The generated application has -O0 settings to enable debugging. Compiling a release version can be enabled using CONFIG_SPEED_OPTIMIZATIONS for the -O2 compiler setting or CONFIG_SIZE_OPTIMIZATIONS for -Os.

6. **Build TFlite static library**: To enable the TFLite static library within the Zephyr project, apply the following commands:

```
$ west config manifest.project-filter \
-- +tflite-micro
$ west config manifest.group-filter \
-- +optional
$ west update
```

   These commands clone the latest version of TFLite static libraries and include them with the Zephyr build environment.

7. **Customize timing functions**: The Agilex 5 SoC FPGA has a 64-bit Arm Cortex-A architecture with 64-bit timers. On the other hand, the TFLite Micro library is based on a 32-bit platform (32-bit timer). To avoid integer overflow, the TFLite Micro time functions must read a 64-bit input from Arm's 64-bit timer.

   Replace the timing function in micro_time.cc as such:

```
#define CLOCK_TICKS_PER_SEC \
400000000
...
uint32_t ticks_per_second() \
{ return CLOCK_TICKS_PER_SEC; }
...
uint64_t GetCurrentTimeTicks() \
{ return (uint64_t)(arch_k_cycle_get_64());
```

In micro_time.h,

```
uint64_t GetCurrentTimeTicks();
...
inline uint64_t TicksToMs(int64_t ticks)
{
    return static_cast<uint64_t>(1000.0f *static_cast<float>(ticks) / static_cast<float>(ticks_per_second()));
}
```

In micro_profiler.h, all the 32-bit tick variables must be changed to 64-bit size.

```
uint64_t start_ticks_[kMaxEvents];
uint64_t end_ticks_[kMaxEvents];
...
struct TicksPerTag {
    const char* tag;
    uint64_t ticks;
};
```

In micro_profiler.cc, any variable that applies the above 32-bit ticks variables must be changed to a 64-bit size.

With that, a newly modified TFLite library and Zephyr OS are compatible with the HPS processor.

8. **Build the application**: The application is ready to build using the following command:

```
$ west build \
-b intel_socfpga_agilex5_socdk samples/\
modules/tflite-micro/hello_world/ \
-d agilex5_tinyml -p
```

Where -b refers to the board used for this build, and -d is used to create the build directory with all the generated files. Inside that directory, there is a sub-directory named zephyr, where the zephyr.bin and zephyr.elf files can be found.

System Deployment and Validation

Once the SoC FPGA completes the SoC boot flow using FSBL and SSBL, you may proceed to download the zephyr.elf file (i.e., TFLite Hello World zephyr application) into the processor core and test the example. Similar results in Figure C will be produced, indicating the successful SoC FPGA platform integration.

Further Enhancements

With the above steps, a model can be quickly deployed to the Nios V or Arm processor during the evaluation stages. In the case of soft processors, custom instructions can be used to deploy the model to accelerate its performance. Most used functions in the model can be profiled and accelerated in the FPGA fabric to improve the overall latency and throughput of the system.

Agilex 5 and Agilex 3 FPGAs have fabric infused with AI Tensor Blocks, giving higher compute density per DSP block of 20 block floating point 16 (Block FP16) or int8 multiply accumulate (MAC) blocks. Other lower precisions, such as FP9 and INT4, are supported in the DSP blocks, which greatly helps accelerate the performance and latency of the most used functions in your model and reduce the model sizes, improving computational efficiencies and memory footprint.

Conclusion

Altera's FPGA portfolio addresses various AI/ML inferencing needs. The Agilex 7, Agilex 5, Agilex 3, Arria 10, Cyclone, and MAX FPGA families can deploy tinyML models and AI/ML at the edge and embedded. At the same time, the Agilex 7 FPGA M-Series can be used for generative AI LLM-based transformer models at the edge or data center.

Agilex 5 FPGAs were the first FPGAs infused with tensor blocks in the fabric itself, giving high compute density, followed by Agilex 3 FPGAs. We also offer high-bandwidth memory (HBM) on Agilex 7 FPGAs M-Series to implement memory-bound models such as LLMs.

The silicon is complemented by easy-to-use software flow tools such as the FPGA AI Suite and OpenVINO™ toolkit for more popular edge applications. They offer a push-button conversation AI model to the FPGA AI Suite inference IP, which can be easily integrated into an FPGA using the Quartus Prime design software.

Integrating tinyML with low-power microcontrollers such as Nios V and Arm processors on Altera FPGAs by leveraging their flexibility and performance enables tinyML to operate on a broad spectrum of devices, bringing intelligent compute to the edge. Users can execute complex AI tasks on smaller, less resource intensive FPGA devices, deploying a cost-effective and power-efficient solution. Users can also implement multiple Nios V processor instances running tinyML on Agilex™ FPGAs. No-cost tools and a unified debugging environment make developing and debugging solutions easy across all Altera SoC FPGAs. Additionally, by porting 32-bit tinyML applications onto the 64-bit Arm Cortex-A architecture, Altera enables the seamless enhancement of computational efficiency and futureproofing embedded ML solutions without any changes to the software.

Users can rapidly innovate by deploying the models on RISC-V or Arm processors in Altera SoC FPGAs. Once the model becomes stable, custom instructions can be used to further accelerate the performance of the model in the case of soft processors.

References

1. Altera® FPGAs and SoCs with FPGA AI Suite and OpenVINO™ Toolkit Drive Embedded/Edge AI/Machine Learning Applications White Paper
2. LiteRT for Microcontrollers
3. TFLite Micro – Porting to a new platform
4. Developing with Zephyr - Getting Started Guide
5. Golden Hardware Reference Design (GHRD) Build Scripts
6. Installing the Ashling RiscFree IDE for Intel FPGAs

altera™
An Intel Company

[1] TensorFlow Lite is renamed as LiteRT on September 04, 2024.