CFU Playground: Full-Stack Open-Source Framework for TinyML Acceleration on FPGAs

Shvetank Prakash†  Tim Callahan‡  Joseph Bushagour§  Colby Banbury*  Alan V. Green†  Pete Warden†  Tim Ansell†  Vijay Janapa Reddi†

The Playground

The custom function unit (CFU) is a small piece of custom logic added in hardware to extend the CPU's datapath to accelerate a discrete function determined by the developer. It follows the RISC-V R-format in which it receives two operands from the register file and writes one result back. A CFU can support state, multiple custom instructions, and pipelining. A notable feature of this architecture is that the CFU does not have direct memory access. It relies on the CPU to move data back and forth. The merits of a direct CPU-memory connection are being considered and may be added in the future. The boundary between CPU and CFU is strictly logical. The implementation flattens the design and optimizes, places, and routes it all together.

CFU Playground runs a complete System-on-Chip (SoC) on an FPGA to capture the full-stack system effects of accelerating ML models. LiteX provides a convenient and efficient infrastructure to create FPGA soft cores and SoCs. The soft core used in CFU Playground is VexRiscv, an implementation of a RISC-V CPU in SpiralHDL. The design of the VexRiscv is highly configurable, providing the ability to easily plugin or remove many different features for performance and functionality such as pipelining stages, caches, and floating point units. This customization ability lends itself well to enabling the design space exploration of CPU vs. CFU.

Hardware

The gateware for CFU Playground is adaptable to a wide range of FPGA hardware platforms. It can fit on a board as small as Fomu, which is 1 cm² and fits entirely within a USB port, which enables rapid prototyping for tinyML. But when more resources are available, a larger more powerful soft CPU and CFU can be built, and with more memory, larger models can be run.

CFU Playground currently supports the Xilinx 7-Series as well as the Lattice iCE40, ECP5, and CrossLink FPGAs. The Fomu with the iCE40UPXS FPGA is close to the smallest usable board; it features 5280 logic cells, 128kB on-chip large RAM, 30 512-byte block RAMs, and 8 16b x 16b DSP/multiplier blocks and is small enough to slot into a USB.

Software

To invoke the CFU, custom instructions must be added to the CPU's instruction set. However, it is not the compiler's responsibility to find uses for the instructions. It only needs to generate them when requested by the user. Therefore, we can use a stock RISC-V GCC toolchain, coupled with a macro we provide that expands to "asm" directives to generate the encoded custom instructions where necessary. The macro takes 4 arguments, and returns one result:

\[ q = \text{cfu-op}(\text{funct7}, \text{funct3}, a, b); \]

The macro directly generates the encoded 32b value, so not even the assembler needs modification. "funct7" and "funct3" are the C/I+ 32b integer variables used as operands for the instruction, and a 32b result is returned.

It is the user's responsibility to call the custom operations from their code. The custom instruction macro can be intermixed with regular C code, similar to any other C/C++ operation. TensorFlow Lite for Microcontrollers (TFLite Micro) is the inference framework that CFU Playground uses for the deployment of the neural network. The user must provide an optimal kernel that uses the new custom instructions to realize the runtime performance improvements.

Evaluation

Image classification is a common task for low-power, always-on cameras, such as those in a smart doobell. We present the experience of one of the authors, with limited prior hardware development experience, utilizing the CFU Playground development environment to optimize —tune— a MobileNetV2 model, which is commonly used for efficient image classification, and optimized its performance on an Arty A7-35T board, which has a Xilinx XCTA35T FPGA with 295 MB of external DDR3 memory. The chart below shows the speedup progressively as we stepped through each of the optimizations. We start with software optimizations and move on to hardware support using the CFU.

Existing Frameworks

<table>
<thead>
<tr>
<th>Source</th>
<th>Stock</th>
<th>Open</th>
<th>Customization</th>
<th>Stock ISA</th>
<th>Custom ISA</th>
<th>Specialized ISA</th>
<th>Compiler</th>
<th>ML Ops</th>
<th>HW acceleration</th>
</tr>
</thead>
</table>