“CFU Playground: Customize Your ML Processor for Your Specific TinyML Model”

Tim Callahan - Google

January 11, 2022
tinyML Talks Strategic Partners

Additional Sponsorships available – contact Olga@tinyML.org for info
Executive Strategic Partners
Arm: The Software and Hardware Foundation for tinyML

1. Connect to high-level frameworks
   - Profiling and debugging tooling such as Arm Keil MDK

2. Supported by end-to-end tooling
   - Optimized models for embedded
     - Runtime (e.g. TensorFlow Lite Micro)
     - Optimized low-level NN libraries (i.e. CMSIS-NN)

3. Connect to Runtime
   - RTOS such as Mbed OS
     - Arm Cortex-M CPUs and microNPUs

AI Ecosystem Partners

Stay Connected

@ArmSoftwareDevelopers
@ArmSoftwareDev

Resources: developer.arm.com/solutions/machine-learning-on-arm

© 2020 Arm Limited (or its affiliates)
The leading edge ML platform

www.edgeimpulse.com
Advancing AI research to make efficient AI ubiquitous

Power efficiency
- Model design, compression, quantization, algorithms, efficient hardware, software tool

Personalization
- Continuous learning, contextual, always-on, privacy-preserved, distributed learning

Efficient learning
- Robust learning through minimal data, unsupervised learning, on-device learning

Perception
- Object detection, speech recognition, contextual fusion

Reasoning
- Scene understanding, language understanding, behavior prediction

Action
- Reinforcement learning for decision making

A platform to scale AI across the industry
SYNTIANT

Neural Decision Processors
- At-Memory Compute
- Sustained High MAC Utilization
- Native Neural Network Processing

ML Training Pipeline
- Enables Production Quality Deep Learning Deployments

Data Platform
- Reduces Data Collection Time and Cost
- Increases Model Performance

End-to-End Deep Learning Solutions for TinyML & Edge AI

Contact: partners@syntiant.com
Website: www.syntiant.com
Platinum Strategic Partners
WE USE AI TO MAKE OTHER AI FASTER, SMALLER AND MORE POWER EFFICIENT

Automatically compress SOTA models like MobileNet to <200KB with little to no drop in accuracy for inference on resource-limited MCUs

Reduce model optimization trial & error from weeks to days using Deeplite’s design space exploration

Deploy more models to your device without sacrificing performance or battery life with our easy-to-use software

BECOME BETA USER bit.ly/testdeeplite
Add Advanced Sensing to your Product with Edge AI / TinyML

Pre-built Edge AI sensing modules, plus tools to build your own

Reality AI solutions

- Prebuilt sound recognition models for indoor and outdoor use cases
- Solution for industrial anomaly detection
- Pre-built automotive solution that lets cars “see with sound”

Reality AI Tools® software

- Build prototypes, then turn them into real products
- Explain ML models and relate the function to the physics
- Optimize the hardware, including sensor selection and placement

https://reality.ai   info@reality.ai   @SensorAI   Reality AI
# Broad and Scalable Edge Computing Portfolio

## Microcontrollers & Microprocessors

<table>
<thead>
<tr>
<th>Arm® Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arm® Cortex®-M 32-bit MCUs</td>
</tr>
<tr>
<td>Arm®-based High-end 32 &amp; 64-bit MPUs</td>
</tr>
<tr>
<td>Arm® Cortex®-M0+ Ultra-low Power 32-bit MCUs</td>
</tr>
<tr>
<td>Renesas Synergy™</td>
</tr>
<tr>
<td>Arm®-based 32-bit MCUs for Qualified Platform</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Renesas Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ultra-low Energy 8 &amp; 16-bit MCUs</td>
</tr>
<tr>
<td>Bluetooth® Low Energy, SubGHz, LoRa®-based Solutions</td>
</tr>
<tr>
<td>High Power Efficiently 32-bit MCUs</td>
</tr>
<tr>
<td>High-resolution HMI, Industrial network &amp; real-time control</td>
</tr>
<tr>
<td>40nm/28nm process Automotive 32-bit MCUs</td>
</tr>
<tr>
<td>Rich functional safety and embedded security features</td>
</tr>
</tbody>
</table>

## Core technologies

- **AI**
  - A broad set of high-power and energy-efficient embedded processors

- **Security & Safety**
  - Comprehensive technology and support that meet the industry’s stringent standards

- **Digital & Analog & Power Solution**
  - Winning Combinations that combine our complementary product portfolios

- **Cloud Native**
  - Cross-platforms working with partners in different verticals and organizations
Gold Strategic Partners
Maxim Integrated: Enabling Edge Intelligence

**Advanced AI Acceleration IC**

The new MAX78000 implements AI inferences at low energy levels, enabling complex audio and video inferencing to run on small batteries. Now the edge can see and hear like never before.

[www.maximintegrated.com/MAX78000](http://www.maximintegrated.com/MAX78000)

**Low Power Cortex M4 Micros**

Large (3MB flash + 1MB SRAM) and small (256KB flash + 96KB SRAM, 1.6mm x 1.6mm) Cortex M4 microcontrollers enable algorithms and neural networks to run at wearable power levels.

[www.maximintegrated.com/microcontrollers](http://www.maximintegrated.com/microcontrollers)

**Sensors and Signal Conditioning**

Health sensors measure PPG and ECG signals critical to understanding vital signs. Signal chain products enable measuring even the most sensitive signals.

[www.maximintegrated.com/sensors](http://www.maximintegrated.com/sensors)
LatentAI

Adaptive AI for the Intelligent Edge

Latentai.com
Build Smart IoT Sensor Devices From Data

SensiML pioneered TinyML software tools that auto generate AI code for the intelligent edge.

- End-to-end AI workflow
- Multi-user auto-labeling of time-series data
- Code transparency and customization at each step in the pipeline

We enable the creation of production-grade smart sensor devices.

sensiml.com
SynSense builds sensing and inference hardware for ultra-low-power (sub-mW) embedded, mobile and edge devices. We design systems for real-time always-on smart sensing, for audio, vision, IMUs, bio-signals and more.

https://SynSense.ai
Silver Strategic Partners
tinyML Summit 2022
Miniature dreams can come true…
March 28-30, 2022
Hyatt Regency San Francisco Airport
https://www.tinyml.org/event/summit-2022/

The Best Product of the Year and the Best Innovation of the Year awards are open for nominations between November 15 and February 28.

tinyML Research Symposium 2022
March 28, 2022
https://www.tinyml.org/event/research-symposium-2022

More sponsorships are available: sponsorships@tinyML.org
Join Growing tinyML Communities:

**tinyML - Enabling ultra-low Power ML at the Edge**


**The tinyML Community**

https://www.linkedin.com/groups/13694488/

7.7k members in 40 Groups in 33 Countries

2.5k members & 4.3k followers
Subscribe to tinyML YouTube Channel for updates and notifications (including this video)
www.youtube.com/tinyML
Next tinyML Talks

<table>
<thead>
<tr>
<th>Date</th>
<th>Presenter</th>
<th>Topic / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuesday, January 18</td>
<td>Ashutosh Pandey, Infineon Technologies</td>
<td>Exploring techniques to build efficient and robust TinyML deployments</td>
</tr>
</tbody>
</table>

Webcast start time is 8:00 am Pacific time

Please contact talks@tinyml.org if you are interested in presenting
Reminders

Slides & Videos will be posted tomorrow

Please use the Q&A window for your questions

tinyml.org/forums   youtube.com/tinyml
Tim Callahan works at Google with the open source FPGA toolchain (SymbiFlow) team. His work is to help make FPGA development more accessible, fun, and rewarding. His research interests include anything that involves optimizing the hardware/software boundary. He has degrees from UC Berkeley, Cambridge University, and the University of Minnesota.
TinyML.org Talk
January 11, 2022
Tim Callahan, tcal@google.com presenting the work of many

Disclaimer: NOT an official Google project

It's on GitHub: Fork it, fix it, send a PR!
Acknowledgements

Googlers: Alan Green, David Lattimore, Dan Callaghan, Tim Ansell, Interns Rachel Sugrono & Joey Bushagour

TFLM (Pete Warden and team)

Antmicro

Harvard: Prof Vijay Reddi, Shvetank Prakash, Colby Banbury

Open source: VexRiscv, LiteX, SymbiFlow, Yosys, Nextpnrr, VTR, Migen, nMigen, Renode, Verilator, OpenOCD, ..
Open Source Showcase!

- ML library: TensorFlow Lite -- open source
- CPU ISA: RISC-V -- open
- CPU design: VexRiscv -- open source
- FPGA SoC/IP: LiteX -- open source
- FPGA synth/PnR: SymbiFlow, Yosys, Nextpnr, VPR -- open source
  - FPGA vendor tools can be used if you wish
- Python HW gen: Migen, nMigen -- open source
- Simulation: Renode, Verilator -- open source

- The only proprietary component is the FPGA itself
Benefits of Open Source

● No licensing fees
● No license headaches in your build system
● You can fix bugs and address shortcomings
● No vendor lock-in
● Transparency -- open for inspection, both sw & hw
  ○ Do you trust a third-party blob of sw or hw integrated with your product?

→ see betrusted.io

“At its core lies not a CPU, but an FPGA, so that users are empowered to build their processors from scratch and know there are no flaws or backdoors in the architecture that could lead to security compromises.”
CFU Playground Key Ideas

- **Specialize** the entire stack for your particular model – both the ML kernels AND the processor

- Make it easy to get started; allow quick iterations → rapid iterative design space exploration
FPGA – programmable hardware

A “bitstream” or “configuration”

- determines the function of each configurable logic block
- and sets switches in the interconnect to connect them to each other and to I/O

Image source: “Parallel Programming for FPGAs”, Ryan Kastner, Janarbek Matai, Stephen Neuendorffer (CC 4.0)
CFU Playground System Architecture

Tensorflow Lite for Microcontrollers

Host Computer

FPGA

RISC V CPU

Custom Function Unit (CFU)

Bus

Serial Port

RAM
CFU Playground -- some of the tested boards
<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>test-results-example_cf u-common_soc-1bitsquared_icebreaker</td>
<td>511 KB</td>
</tr>
<tr>
<td>test-results-example_cf u-common_soc-antmicro_lppdr4_test_board</td>
<td>511 KB</td>
</tr>
<tr>
<td>test-results-example_cf u-common_soc-camlink_4k</td>
<td>512 KB</td>
</tr>
<tr>
<td>test-results-example_cf u-common_soc-colorlight_i5</td>
<td>511 KB</td>
</tr>
<tr>
<td>test-results-example_cf u-common_soc-decklink_mini_4k</td>
<td>511 KB</td>
</tr>
<tr>
<td>test-results-example_cf u-common_soc-decklink_quad_hdmi_recorder</td>
<td>511 KB</td>
</tr>
<tr>
<td>test-results-example_cf u-common_soc-digitentl_arty</td>
<td>511 KB</td>
</tr>
<tr>
<td>test-results-example_cf u-common_soc-digitentl_arty_s7</td>
<td>511 KB</td>
</tr>
<tr>
<td>test-results-example_cf u-common_soc-digitentl_genesys2</td>
<td>511 KB</td>
</tr>
</tbody>
</table>
CFU Playground Prerequisites:

- Linux host

- You will use:
  - git
  - C++ (mostly C)
  - Hardware design – Verilog or ???
    - unless you just want to experiment with CPU configuration

- A supported FPGA board
  - or you could just simulate with Renode or Verilator
CFU Playground Uses

- **Deploy** a soft CPU+CFU on FPGA for tinyML
  - Firmware updates include a new ML model, new software, and a new CPU+CFU

- **Prototype** a custom RISC-V-based ASIC
  - Yes you! efabless.com MPW will fab your open-source chip

- **Learn** about ML software, hardware, and performance

- **Research** new ML approaches
  - while co-designing the hardware to support it
The Hardware Lottery

- Sara Hooker’s observation that the success of new ML approaches depends on their compatibility with downstream software and hardware
- Here you can “make your own luck”!
RISC-V and CFU Basics
The RISCV Add Instruction

00720C33  add  x24, x4, x7  #  x24 = x4 + x7
The RISCV Add Instruction

```
00720C33  add  x24,x4,x7  # x24 = x4 + x7
```

0000000  00111  00100  000  11000  0110011
## The RISCV Add Instruction

<table>
<thead>
<tr>
<th>0000000</th>
<th>00111</th>
<th>00100</th>
<th>000</th>
<th>11000</th>
<th>0110011</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>ALU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Binary Code and Instruction

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00720C33</td>
<td>add x24, x4, x7</td>
</tr>
</tbody>
</table>

# x24 = x4 + x7
<table>
<thead>
<tr>
<th>0000000</th>
<th>00111</th>
<th>00100</th>
<th>000</th>
<th>11000</th>
<th>0110011</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ALU</td>
</tr>
</tbody>
</table>

**Register File**

**ALU**
<table>
<thead>
<tr>
<th>funct7</th>
<th>rs2</th>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>x7</td>
<td>x4</td>
<td>ADD</td>
<td>x24</td>
<td>ALU</td>
</tr>
</tbody>
</table>

```
0000000 00111 00100 000 11000 0110011
```

Diagram:
- Register File
- ALU
```
<table>
<thead>
<tr>
<th>funct7</th>
<th>rs2</th>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>x7</td>
<td>x4</td>
<td>ADD</td>
<td>x24</td>
<td>ALU</td>
</tr>
</tbody>
</table>
```

![Diagram of register file and ALU connections]
The diagram illustrates the operation of an ALU and a Register File. The instruction format is shown in the table:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>00111</td>
<td>00100</td>
<td>000</td>
<td>11000</td>
<td>0110011</td>
<td></td>
</tr>
<tr>
<td>funct7</td>
<td>rs2</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>x7</td>
<td>x4</td>
<td>ADD</td>
<td>x24</td>
<td>ALU</td>
<td></td>
</tr>
</tbody>
</table>

The diagram shows the flow of data from the Register File to the ALU, with arrows indicating the paths of rs1, rs2, funct3, and funct7.
<table>
<thead>
<tr>
<th>funct7</th>
<th>rs2</th>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>x7</td>
<td>x4</td>
<td>ADD</td>
<td>x24</td>
<td>ALU</td>
</tr>
<tr>
<td>funct7</td>
<td>rs2</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>ADD</td>
<td>x7</td>
<td>x4</td>
<td>ADD</td>
<td>x24</td>
<td>ALU</td>
</tr>
<tr>
<td>CFUOP</td>
<td>rs2</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>--------</td>
<td>-----</td>
<td>--------</td>
</tr>
<tr>
<td>x7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The image shows a table with columns for CFUOP, rs2, rs1, funct3, rd, and opcode, along with a diagram illustrating connections between the Register File, ALU, and CFU.
```
0000000 00111 00100 000 11000 0001011
       funct7    rs2    rs1    funct3    rd    opcode
CFUOP  x7       x4       CFUOP   x24       CUSTOM
```

Diagram:
- Register File
  - x4 → ALU
  - x7 → ALU
- ALU
  - funct3 CFUOP
- CFU
  - rs1 → CFU
  - rs2 → CFU
  - funct7 CFUOP
### Register File

- **Register File**
  - Inputs: x4, x7
  - Outputs: x24

### ALU

- **ALU**
  - Inputs: funct3 CFUOP, funct7 CFUOP

### CFU

- **CFU**
  - Inputs: rs1, rs2, funct7 CFUOP, funct3 CFUOP
  - Outputs: rd

### Opcode Table

<table>
<thead>
<tr>
<th>Opcode</th>
<th>rs2</th>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000</td>
<td>00111</td>
<td>00100</td>
<td>000</td>
<td>11000</td>
<td>0001011</td>
</tr>
<tr>
<td>funct7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFUOP</td>
<td>x7</td>
<td>x4</td>
<td>CFUOP</td>
<td>x24</td>
<td>CUSTOM</td>
</tr>
</tbody>
</table>

- **CFUOP**
  - Controls: x7, x4, x24
  - Duties: CUSTOM
<table>
<thead>
<tr>
<th></th>
<th>funct7</th>
<th>rs2</th>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFUOP</td>
<td>x7</td>
<td>x4</td>
<td>CFUOP</td>
<td>x24</td>
<td></td>
<td>CUSTOM</td>
</tr>
</tbody>
</table>

```
0000000 00111 00100 00 11000 0001011
```
CFU details

- Two operands from the regfile, one result written back
- Multiple/variable cycles; can refuse new inputs while working → pipelined or iterative computation
- NO direct connection between CFU & the memory hierarchy
- CFU cannot otherwise affect CPU state (branch etc.)
- CFU can contain state -- registers and memories that persist
- CFU can contain its own sequencer
- One CFU, multiple instructions that can access the shared state
So how do we use the CFU?

- Do we build a nifty compiler? NO!
- YOU are the compiler..it’s up to you to insert uses of your new instructions into the code
- After all, you’re the one who just designed the new instruction to speed up your code, so you know exactly where it will be used
How to use the CFU from software

- Access the new instruction using function call syntax:

```c
rslt = cfu_op(funct3, funct7, op1, op2);
```

- Compile-time constants
- C / C++ variables / expressions
How to use the CFU from software

- Access the new instruction using function call syntax:

  ```c
  rslt = cfu_op(funct3, funct7, op1, op2);
  ```

  ```c
  rslt = cfu_op(0, 1, op1, op2);
  ```

Compile-time constants

C / C++ variables / expressions
How to use the CFU from software

- Access the new instruction using function call syntax:

  \[
  rslt = \text{cfu-op}(\text{funct3, funct7, op1, op2});
  \]

```c
#define HAMMING_DISTANCE(x,y) \text{cfu-op}(0, 1, (x), (y));
```

```c
rslt = \text{HAMMING\_DISTANCE}(\text{op1, op2});
```
How to use the CFU from software

- Custom instruction macros intermix with regular C code:

  \begin{align*}
  t1 &= *x; \\
  t2 &= \text{cfu\_op}(0, 0, t1, b); \\
  t3 &= \text{cfu\_op}(1, 0, t2, b); \\
  *x &= t3;
  \end{align*}

  Compiled and disassembled:

  \begin{verbatim}
  400001a0:       00812783            lw        a5,8(sp)
  400001a4:       00d7878b            cfu[0,0]  a5, a5, a3
  400001a8:       00d7978b            cfu[0,1]  a5, a5, a3
  400001ac:       00f12423            sw        a5,8(sp)
  \end{verbatim}

  Objdump can’t disassemble the custom instructions, 
  so we wrote a helper script..

  \textbf{No overhead!}
Why use a CFU approach for tinyML?

- With ML inference, the hot spots are small & very hot
- A SMALL amount of custom hardware to exploit the bit-level flexibility of FPGA can accelerate a LARGE fraction of the runtime
- Leave complexity, setup, outer loops in SW
- Iterative approach: easy to take first step, then next step, ...
- In our experience, the CFU will incrementally grow to become almost a full-blown “accelerator”
TensorFlow Lite for Microcontrollers (TFLM)
TensorFlow Lite for Microcontrollers

All instances use the same kernel...

...but a kernel can dispatch to a specialized version

```cpp
Conv2D(.....)
{
  if (filter.shape() is 1x1xC) {
    Conv2D_1x1(....)
    return;
  }
  for ( ... ) {
    for ( ... ) {
      for ( ... ) {
        ....
      }
    }
  }
}
```
CFU Playground
CFU Playground Build

$ cd proj/proj_template_v/  # establish project

$ make prog 3.5min  # build gateware using ./cfu.v and put the bitstream on the FPGA

$ make -j8 software <1min  # build software w/ local overrides

$ make load  # load and execute the software

...then interact with the software running on the board
Hello, World!

CFU Playground
---------------
1: TfLM Models menu
2: Functional CFU Tests
3: Project menu
4: Performance Counter Tests
5: TFLite Unit Tests
6: Benchmarks
7: Util Tests
main> 1

Running TfLM Models menu

TfLM Models
-----------
1: Person Detection int8 model
2: Micro Speech
3: MLCommons Tiny V0.1 Keyword Spotting
x: exit to previous menu
models> 3

Tests for kws model
-------------------
0: Run with "down" input
1: Run with "go" input
2: Run with "left" input
g: Run golden tests (check for expected outputs)
x: exit to previous menu
kws> g

Running Run golden tests (check for expected outputs)

............
"Event","Tag","Ticks"
0,CONV_2D,19841
1,DEPTHWISE_CONV_2D,4306
2,CONV_2D,11690
3,DEPTHWISE_CONV_2D,4226
4,CONV_2D,11662
5,DEPTHWISE_CONV_2D,4230
6,CONV_2D,11050
7,DEPTHWISE_CONV_2D,4757
8,CONV_2D,11905
9,AVERAGE_POOL_2D,237
10,RESHAPE,3
11,FULLY_CONNECTED,16
12,SOFTMAX,26
Performance counters not enabled.
86M (85996663) cycles total
OK Golden tests passed
CFU Playground Development

- Start a new project by copying a template:
  - `cp -r proj/proj_template proj/my_first_project`
- Choose or bring the TFLite model
- Use built-in profiling to identify time-consuming TF ops
- Look at tensor shapes and other parameters, find opportunities to specialize/simplify the kernel(s)
- Then, look for CFU acceleration opportunities
  - Design CFU, alter TFLM kernels to use the custom instructions
- Measure speedup, repeat!
Examples
Simple Example

This Conv2D kernel consumes 76% of the execution time with model “pdti8”, so go after the computation in the innermost loop.
Simple Example

\[ \text{acc} += \text{filter\_val} \times (\text{input\_val} + \text{input\_offset}) \]
Simple Example

```
// The CFU computation

reg signed [31:0] input_offset; // state
reg signed [31:0] acc;        // state
wire signed [31:0] filt_val = in1;
wire signed [31:0] input_val = in2;
wire signed [31:0] newacc = acc + (filt_val * (input_val + input_offset));
assign out = acc;

always @(posedge clk) begin
    if (cmd_valid) begin
        if (funct3 == 3’b000) begin
            input_offset <= in1;
        end else if (funct3 == 3’b001) begin
            acc <= in1;
        end else if (funct3 == 3’b010) begin
            acc <= newacc;
        end
    end
end
```

Verilog!

\[
\text{acc += filter\_val \ast (input\_val + input\_offset)}
\]
Simple Example

const int32_t input_offset = params.input_offset;  // r = s(q - Z)
// CFU: copy input_offset into the CFU
cfu_op(0, 0, input_offset, 0);

for (int batch = 0; batch < batches; ++batch) {
  for (int out_y = 0; out_y < output_height; ++out_y) {
    const int in_y_origin = (out_y * stride_height) - pad_height;
    for (int out_x = 0; out_x < output_width; ++out_x) {
      const int in_x_origin = (out_x * stride_width) - pad_width;
      for (int out_channel = 0; out_channel < output_depth; ++out_channel) {
          // int32_t acc = 0;
          // CFU: set the CFU internal acc to ZERO
          cfu_op(1, 0, 0, 0);

          for (int filter_y = 0; filter_y < filter_height; ++filter_y) {
            const int in_y = in_y_origin + dilation_height_factor * filter_y;
            for (int filter_x = 0; filter_x < filter_width; ++filter_x) {
              const int in_x = in_x_origin + dilation_width_factor * filter_x;

                for (int in_channel = 0; in_channel < input_depth; ++in_channel) {
                  int32_t input_val = input_data[Offset(input_shape, batch, in_y, in_x, in_channel)];
                  int32_t filter_val = filter_data[Offset(filter_shape, out_channel, filter_y, filter_x, in_channel)];

                  // acc += filter_val * (input_val + input_offset);
                  // CFU: add-multiply-accumulate in the CFU
                  cfu_op(2, 0, filter_val, input_val);
                }
          }
      }
    }
  }
}

// CFU: retrieve final acc value from the CFU
int32_t acc = cfu_op(3, 0, 0, 0);

31% cycle reduction for CONV_2D
24% cycle reduction overall
(model “pdti8”)
Simple Example

Add aliases for the custom instructions, for readability

31% cycle reduction for CONV_2D
24% cycle reduction overall (model “pdti8”)
Typical CFU evolution

acc += filter_val * (input_val + input_offset)
Typical CFU evolution

\[ \text{acc} += \text{filter\_val} \times (\text{input\_val} + \text{input\_offset}) \]
Typical CFU evolution

```
acc += filter_val * (input_val + input_offset)
```
Typical CFU evolution

And after a few more iterations, you’ve added:

- Local memories for both input_vals and filter_vals
- Increased MACC parallelism
- Sequencer for complete matrix x vector computation
- Activation function & scaling
- Double buffering at input and output buffers to pipeline data transfer and computation
Typical CFU evolution

1. Load filter values
2. Set up sequencer
3. Stream input values
4. Stream output values
MobileNet v2

Before speeding up conv_2d (1x1):

Totals

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Count</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFTMAX</td>
<td>11510</td>
<td>0.0M</td>
</tr>
<tr>
<td>RESHAPE</td>
<td>21887</td>
<td>0.0M</td>
</tr>
<tr>
<td>FULLY_CONNECTED</td>
<td>48284</td>
<td>0.0M</td>
</tr>
<tr>
<td>AVERAGE_POOL_2D</td>
<td>487497</td>
<td>0.5M</td>
</tr>
<tr>
<td>ADD</td>
<td>4564330</td>
<td>4.6M</td>
</tr>
<tr>
<td>MUL</td>
<td>7236662</td>
<td>7.2M</td>
</tr>
<tr>
<td>SUB</td>
<td>16517877</td>
<td>16.5M</td>
</tr>
<tr>
<td>CONV_2D_3x3</td>
<td>95241303</td>
<td>95.2M</td>
</tr>
<tr>
<td>DEPTHWISE_CONV_2D</td>
<td>197214007</td>
<td>197.2M</td>
</tr>
<tr>
<td>CONV_2D_1x1</td>
<td>559817289</td>
<td>559.8M</td>
</tr>
</tbody>
</table>
MobileNet v2

55x speedup!
(on just conv2d 1x1)

But Amdahl’s law...
overall speedup is just 2.8x
**MobileNet v2**

After speeding up conv_2d (1x1):

Totals

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Count</th>
<th>(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFTMAX</td>
<td>11503</td>
<td>(0.0M)</td>
</tr>
<tr>
<td>RESHAPE</td>
<td>21886</td>
<td>(0.0M)</td>
</tr>
<tr>
<td>FULLY_CONNECTED</td>
<td>50183</td>
<td>(0.1M)</td>
</tr>
<tr>
<td>AVERAGE_POOL_2D</td>
<td>494322</td>
<td>(0.5M)</td>
</tr>
<tr>
<td>ADD</td>
<td>4577562</td>
<td>(4.6M)</td>
</tr>
<tr>
<td>MUL</td>
<td>7233115</td>
<td>(7.2M)</td>
</tr>
<tr>
<td>CONV_2D_1x1</td>
<td>10263213</td>
<td>(10.3M)</td>
</tr>
<tr>
<td>SUB</td>
<td>16517493</td>
<td>(16.5M)</td>
</tr>
<tr>
<td>CONV_2D_3x3</td>
<td>76621863</td>
<td>(76.6M)</td>
</tr>
<tr>
<td>DEPTHWISE_CONV_2D</td>
<td>199485469</td>
<td>(199.5M)</td>
</tr>
</tbody>
</table>

Originally 560M
Example #2
Keyword Spotting on Fomu
About my Intern Joey Bushagour’s project

- Adding support for tiny FPGAs in CFU-Playground
- Fitting TensorFlow Lite for Microcontrollers on tiny FPGAs
- Making inference on tiny FPGAs faster

fomu.im
75x speedup on model inference

How it started:

```
Running MLCommons Tiny V0.1 Keyword Spotting
Error reporter OK!
Input: 490 bytes, 4 dims: 1 49 10 1

Tests for kws model
---------------------
0: Run with "down" input
1: Run with "go" input
2: Run with "left" input
g: Run golden tests (check for expected outputs)
x: eXit to previous menu
kws>
```

How it's going:

```
Running MLCommons Tiny V0.1 Keyword Spotting
Error reporter OK!
Input: 490 bytes, 4 dims: 1 49 10 1

Tests for kws model
---------------------
0: Run with "down" input
1: Run with "go" input
2: Run with "left" input
g: Run golden tests (check for expected outputs)
x: eXit to previous menu
kws>
```
Optimization: Use CFU SIMD MAC in TFLM ops

- Convolution and depthwise convolution are mostly multiply and accumulate
- Sometimes these 8 bit multiply and accumulates are contiguous in memory
- Our registers are 32 bits wide, we can vectorize where possible

Cumulative speedup: **32.10x**
What's left

- We have 75× speedup
- If we could find room to fit a branch predictor and bypass we'd have 102×
- Optimizing depthwise convolution more

Running MLCommons Tiny V0.1 Keyword Spotting
Error_reporter OK!
Input: 490 bytes, 4 dims: 1 49 10 1

Tests for kws model
------------------------
0: Run with "down" input
1: Run with "go" input
2: Run with "left" input
g: Run golden tests (check for expected outputs)
x: eXit to previous menu
kws>
Wrap up
Generating hardware from Python?

- Yes! We used nMigen (now “Amaranth HDL”) to build our large CFUs.
- It is a Python library for generating HW
- You still need to understand HW
- But you can use Python conveniences (functions, loops, dictionaries, etc)
- Google “pyconline au 2021 cfu” to find Alan’s presentation on YouTube.
Join In The Fun

Clone it: [github.com/google/CFU-Playground](https://github.com/google/CFU-Playground)
- plenty of sample code and models to accelerate
- works with many LiteX supported boards ([check the wiki!](https://github.com/google/CFU-Playground/wiki))

Docs: [cfu-playground.readthedocs.io](https://cfu-playground.readthedocs.io)
- introductions to nmigen, step-by-step guides, detailed documentation

Contact us:
- raise a github issue
- mail us: tcal@google.com, avg@google.com
- chat us: [https://gitter.im/CFU-Playground/community](https://gitter.im/CFU-Playground/community)
End
Copyright Notice

This multimedia file is copyright © 2022 by tinyML Foundation. All rights reserved. It may not be duplicated or distributed in any form without prior written approval.

tinyML® is a registered trademark of the tinyML Foundation.

www.tinyml.org
Copyright Notice

This presentation in this publication was presented as a tinyML® Talks webcast. The content reflects the opinion of the author(s) and their respective companies. The inclusion of presentations in this publication does not constitute an endorsement by tinyML Foundation or the sponsors.

There is no copyright protection claimed by this publication. However, each presentation is the work of the authors and their respective companies and may contain copyrighted material. As such, it is strongly encouraged that any use reflect proper acknowledgement to the appropriate source. Any questions regarding the use of any materials presented should be directed to the author(s) or their companies.

tinyML is a registered trademark of the tinyML Foundation.

www.tinyML.org