

EMBEDDED AI ON L31 CORE

COMPACT NEURAL NETWORK ACCELERATOR IN CODAL

DOMAIN	Neural networks, AI accelerators, handwriting recognition (MNIST)
PRODUCT	Codalip L31™ processor, Codalip Studio™
RESULT	Custom L31 core with AI accelerator
PUBLISHED	September 2022

1 The Context

Recent years of IoT/IloT evolution resulted in an important shift from cloud-level to device-level AI processing. It enables the devices to run some AI tasks locally thus minimizing the security issues, data transfer costs and latency. The ability to run AI/ML tasks becomes a must-have when selecting an SoC or MCU for IoT and IloT applications.

Embedded devices are typically resource-constrained, making it difficult to run AI algorithms on embedded platforms. The Codalip Application Engineering team looked at what could make it easier from a software and hardware point of view. They used the **Codalip L31 RISC-V core** and **Codalip Studio** to explore and customize the design with the efficient and compact AI accelerator tightly coupled with CPU pipeline.

2 The Scope of the project

The Codalip Application Engineering team used TensorFlow Lite for Microcontrollers (TFLite-Micro) as a dedicated AI framework and compared the performance of the Codalip L31 processor core with both standard ISA and custom extensions that include AI accelerator instructions. The accelerator was implemented with less than 200 lines of CodAL code, providing **>5x** performance increase and **>3x** less energy consumption. This project highlighted the benefits of customized design flexibility provided by Studio tools and CodAL simplicity. All these ultimately boost the neural network performance on embedded systems.

3 The Development

The team used TensorFlow Lite Micro with a Cudasip L31 RISC-V core to run a convolutional neural network for image classification. The neural network architecture contains two convolutional and pooling layers, at least one fully-connected layer, vectorized nonlinear functions, data resize, and normalization operations (Figure 1).

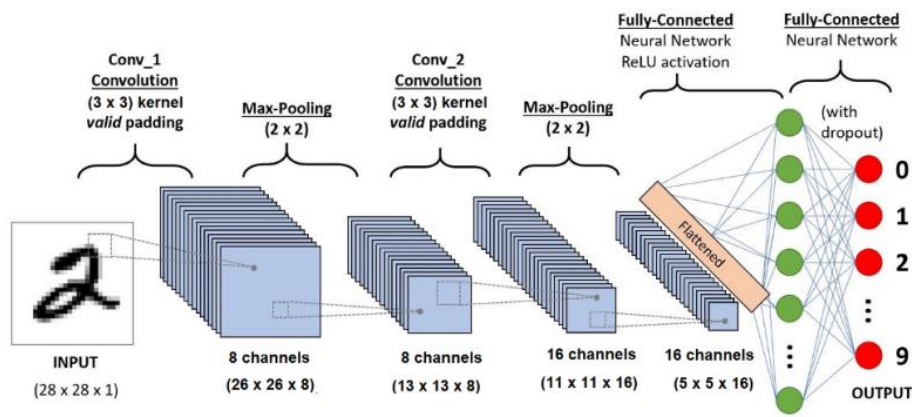


Figure 1 Convolutional neural network architecture

The team took the well-known “MNIST handwritten digits classification” benchmark and used the **Cudasip Studio profiler** (Figure 2) to analyze the image classification task. Cudasip Studio makes it easy to see which code parts are critical, taking the most processor’s time and being primary candidates for optimization.

Symbol	Address	Instructions	Instructions Percent	Cycles	Cycles Percent
tflite::reference_integer_ops::ConvPerChannel	36fa6	6572379	86.3 %	9340321	83.9 %
tflite::reference_integer_ops::MaxPool	45e60	412255	6.4 %	710898	6.4 %
tflite::reference_integer_ops::FullyConnected	3e388	158370	2.1 %	236154	2.1 %

Figure 2 Cudasip Studio Profiler report

Benchmark profiling shows that ~84% of the cycles were spent on the image convolution function, which is implemented by deeply nested for-loops. To make a simple 3x3 convolution, the general-purpose RISC-V processor has to run 9 load instructions, 9 multiplications, 8 additions and there is some overhead for pipeline stalling. Therefore, the convolution seems to be a major candidate for acceleration.

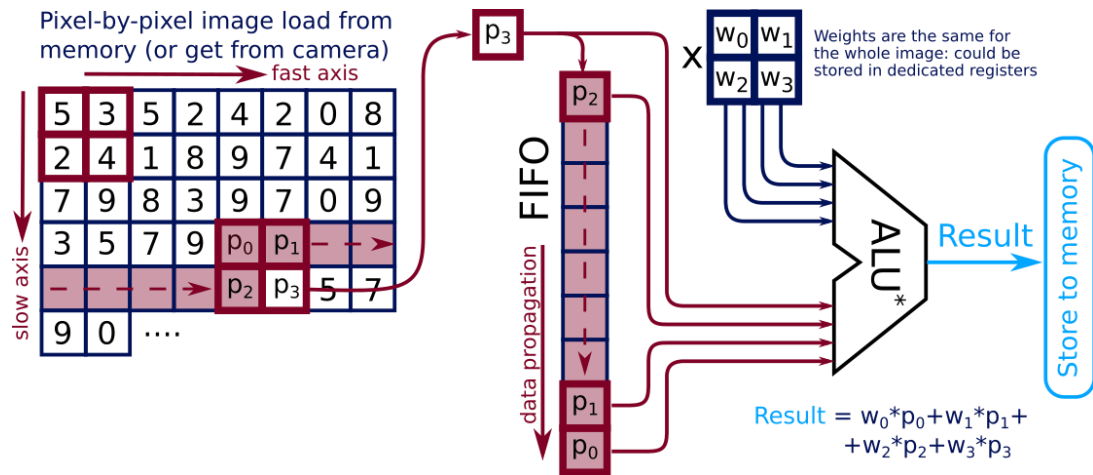





Figure 3 Convolution accelerator principle

The proposed convolution accelerator scheme is shown in Figure 3. It is tightly coupled to the core pipeline and is based on a FIFO register chain that can store multiple image lines. By pushing the image pixels to the FIFO, and reading certain FIFO elements, one can access all image data required for a single convolution in a single processor clock cycle, thus avoiding multiple loads from the memory. The modified ALU* performs parallel multiplications of the image pixels by *convolution weights* and sums up the result. This accelerator, controlled by a couple of custom instructions completes a convolution result in each clock cycle, thus significantly reducing the time required for the entire image convolution.

The described accelerator can certainly be implemented with industry-standard hardware-description languages, however CodAL provides a much simpler and compact way to do that.







```

module rf_gpr #(parameter xlen = 64, parameter size = 32,
parameter resetval = 32'b0, localparam aw = $clog2(size))
( input wire clk, input wire rst, input wire w0_we,
input wire [aw-1:0] w0_wa, input wire [xlen-1:0] w0_d,
input wire r0_re, input wire [aw-1:0] r0_ra,
output wire [xlen-1:0] r0_q, input wire r1_re,
input wire [aw-1:0] r1_ra, output wire [xlen-1:0] r1_q );
reg [xlen-1:0] mem[size-1:0];
integer i;

always @(posedge clk or negedge rst)
if (~rst) begin
for (i = 0; i < size; i = i + 1)
mem[i] <= resetval;
end else if (w0_we) begin
mem[w0_wa] <= w0_d;
end

assign r0_q = r0_re ? mem[r0_ra] : (xlen)'(0);
assign r1_q = r1_re ? mem[r1_ra] : (xlen)'(0);
endmodule

```

```

arch register_file bit[32] rf_gpr
{
dataport r0, r1 {flag = R;};
dataport w0 {flag = W;};
size = 32;
reset = true;
default = 0;
};

```

CodAL provides many constructs facilitating standard processor features design:

- Register files
- Memories (cache, TCM)
- On-chip debugger, trace, etc

Many tasks are automated when using CodAL.

- Automatic modules interconnect, decoder generation

Figure 4 CodAL compactness versus industry-standard HDL (Verilog).

CodAL is a hardware description language designed specifically for processor description. It helps to simplify the definition of some standard processor features such as register files, caches, tightly-coupled memories, and on-chip debuggers. The resulting CodAL representation is much more compact than what other HDLs would provide (Figure 4).

Thanks to this compactness, the convolution accelerator described above can be implemented in CodAL with only **200 lines of code**. It includes both the hardware resource coverage and custom control instructions implementation. The resulting accelerator design can work with up to **64 x 64** images and makes up to **5 x 5** convolutions. The compiler and profiler are automatically generated, which simplifies further the evaluation of the accelerator's performance.

4 The Result

Adding a convolution accelerator and custom instructions to control it led to a custom L31 core with better performance and total energy consumption than the standard L31.

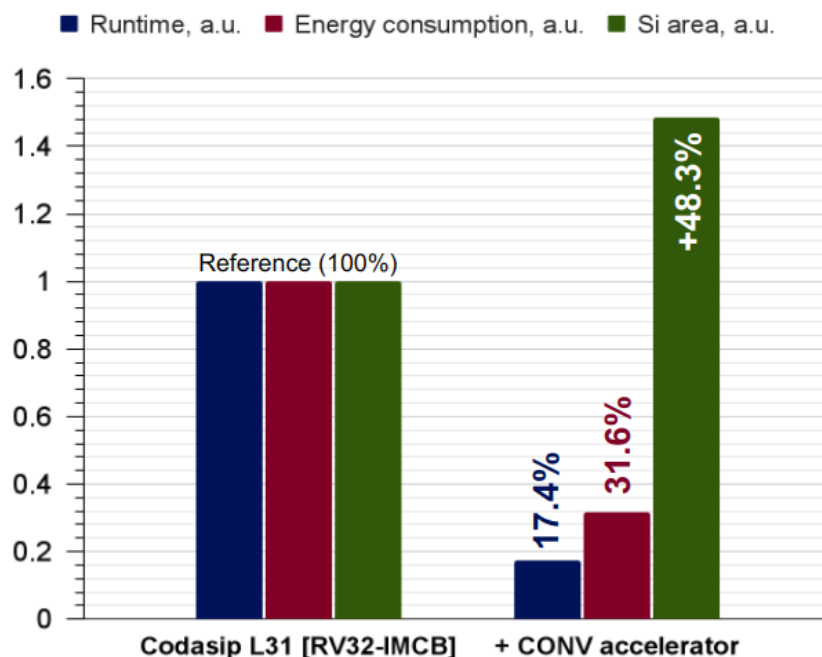


Figure 4 The benefits of adding custom instructions

The number of clock cycles required for image classification was reduced by more than **5 times** and the energy consumption reduced by more than **3 times**. The latter one can be directly related with the battery lifetime, meaning **3 times more** image classifications

between recharge cycles can be done. The improved performance and energy consumption come with the hardware cost of an additional **48%** of silicon area. This is due to parallel multipliers inside the convolution accelerator module, and it is considered to be a reasonable cost for the significant performance and energy gains.

Note: AI & ML applications vary in their computational requirements. The custom design example provided above is given for illustration purposes only and does not pretend to be a complete and optimized solution. Other custom instructions and/or accelerator designs might result in further PPA improvements.

About Codasip

Codasip delivers leading-edge RISC-V processor IP and high-level processor design tools, providing IC designers with all the advantages of the RISC-V open ISA, along with the unique ability to customize the processor IP. As a founding member of RISC-V International and a long-term supplier of LLVM and GNU-based processor solutions, Codasip is committed to open standards for embedded and application processors. Formed in 2014 and headquartered in Munich, Germany, Codasip currently has R&D centers in Europe and sales representatives worldwide. For more information about our products and services, visit www.codasip.com.