2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

# Compact AI accelerator for embedded applications

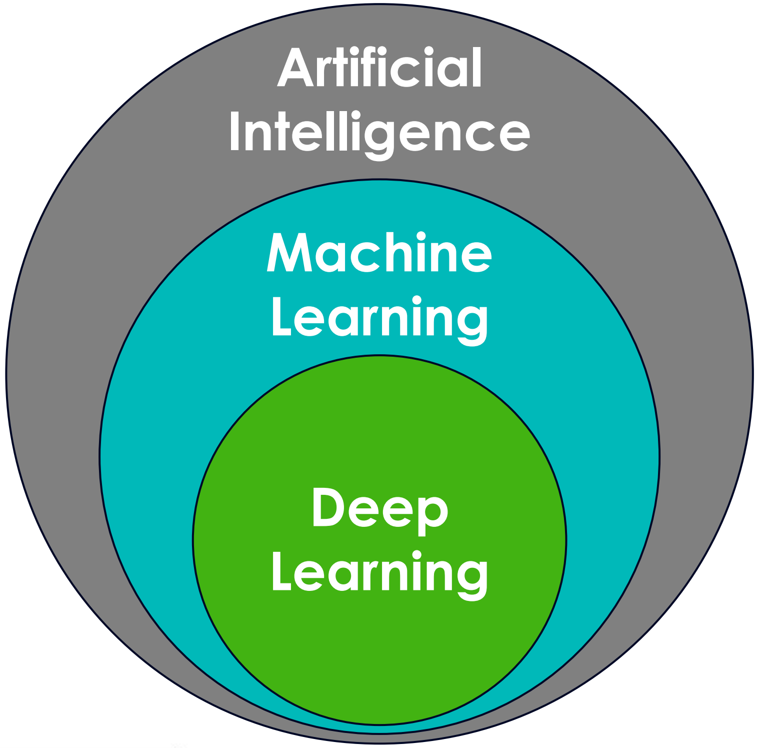
Alexey Shchekin, Solutions Engineer

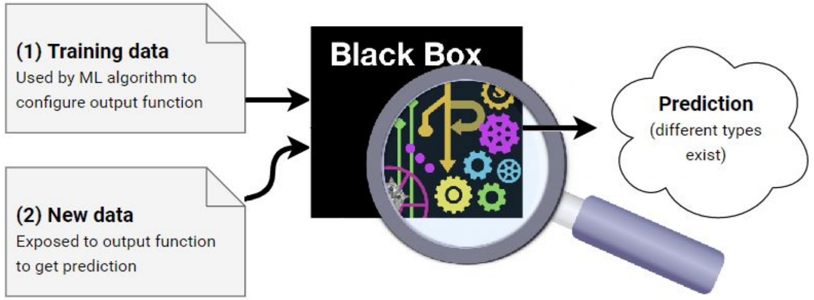
Codasip s.r.o.

codasip®

accellera SYSTEMS INITIATIVE

# Embedded AI and custom HW challenges




(1) Training data
Used by ML algorithm to configure output function
(2) New data
Exposed to output function to get prediction
Black Box
Prediction
(different types exist)

## Embedded AI: artificial intelligence at the device level

Strong tendency to off-load cloud computing and run some simple AI tasks on-device:

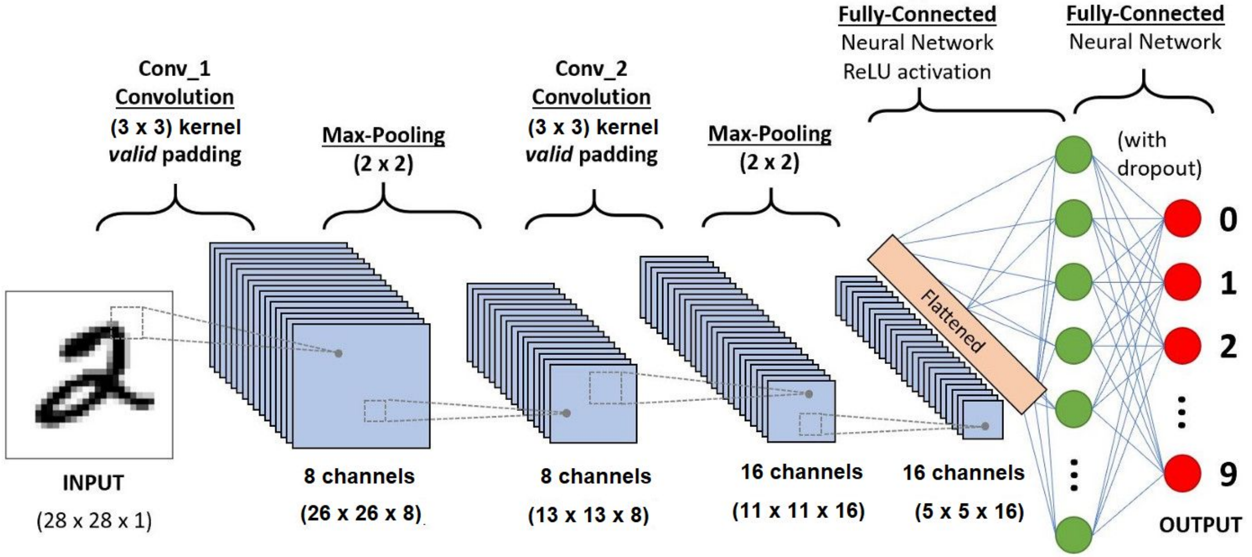
- **Autonomy**: minimize data transfer and related energy consumption
- **Security**: finish sensitive data processing locally
- **Modular device design** with embedded AI SoCs
- **Reduced latency** for critical IoT/IIoT infrastructure

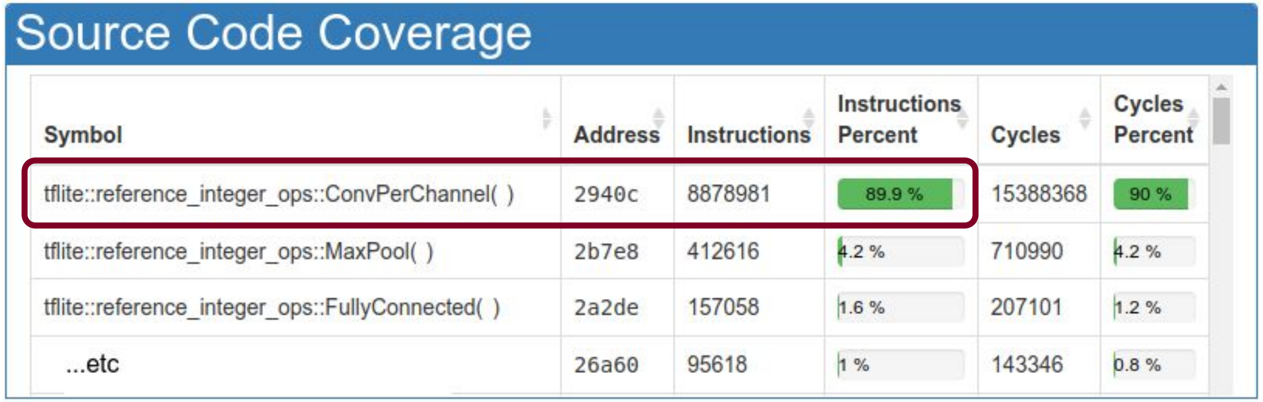
## Requirements & challenges

3 cornerstones of AI:

- **Model & Algorithm**
  => frameworks: Tensorflow, PyTorch, Caffe, etc
- **Data, lots of data**
  => open datasets: Kaggle etc
- **Computing power**
  => *custom-designed accelerators* that soften embedded resource constraints (small memory, limited ISA, lack of DSP)

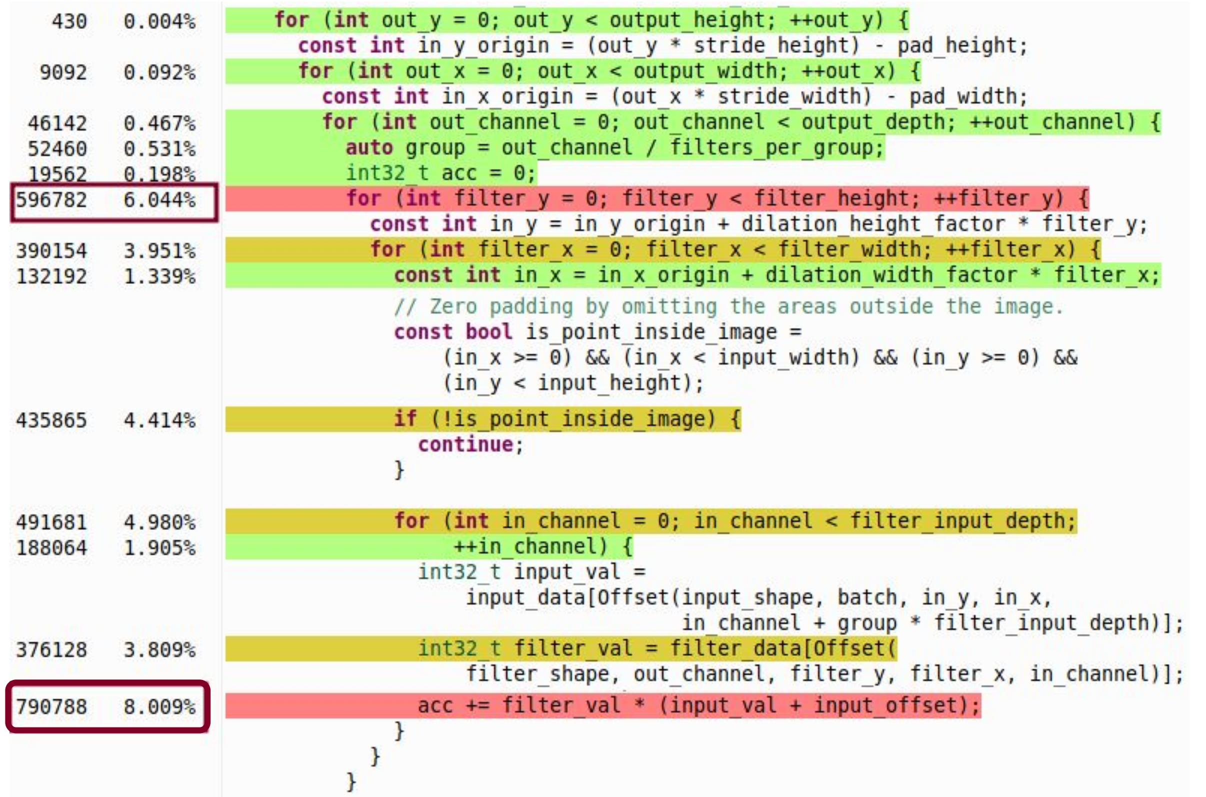
# Profiling TFLite Image classification model





## Source Code Coverage

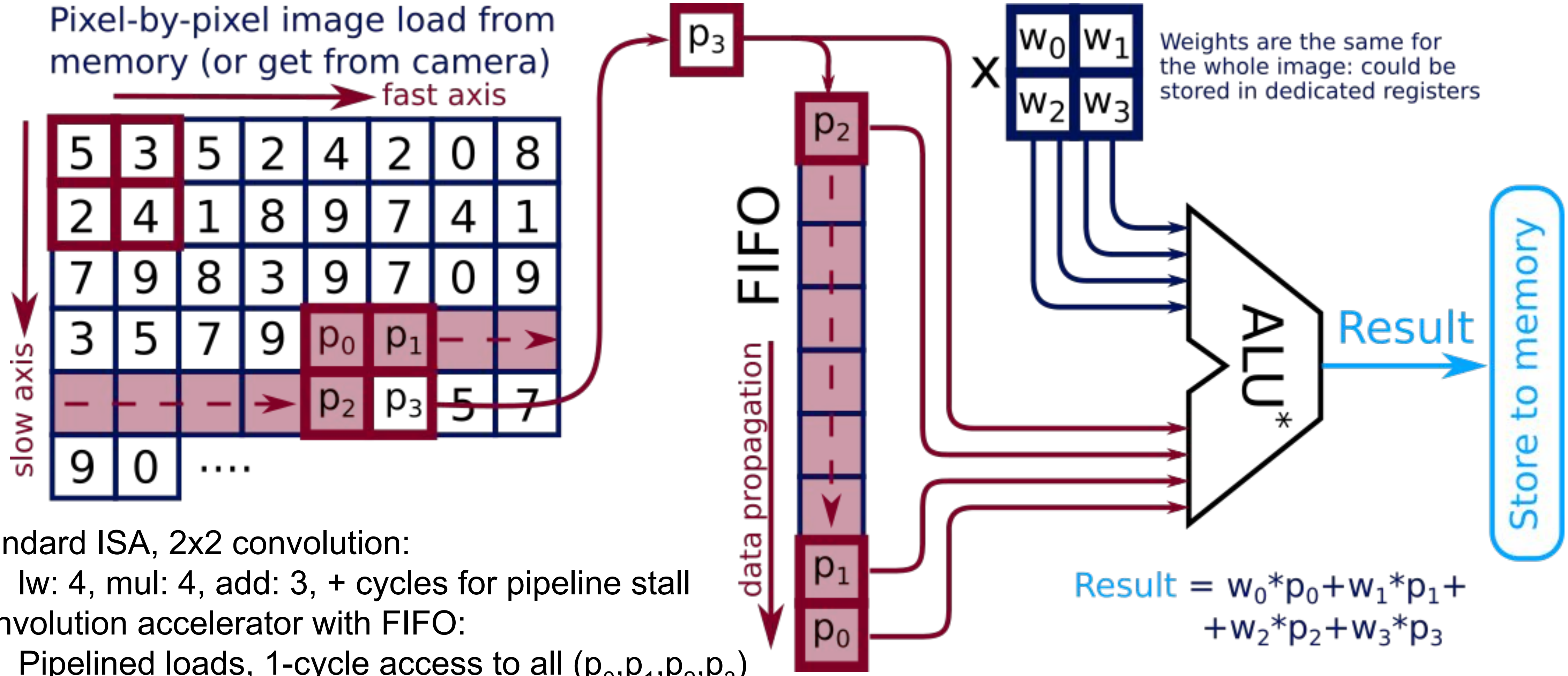
| Symbol | Address | Instructions | Instructions Percent | Cycles | Cycles Percent |
|---|---|---|---|---|---|
| tflite::reference_integer_ops::ConvPerChannel( ) | 2940c | 8878981 | 89.9 % | 15388368 | 90 % |
| tflite::reference_integer_ops::MaxPool( ) | 2b7e8 | 412616 | 4.2 % | 710990 | 4.2 % |
| tflite::reference_integer_ops::FullyConnected( ) | 2a2de | 157058 | 1.6 % | 207101 | 1.2 % |
| ...etc | 26a60 | 95618 | 1 % | 143346 | 0.8 % |



```
   430   0.004%    for (int out_y = 0; out_y < output_height; ++out_y) {
                     const int in_y_origin = (out_y * stride_height) - pad_height;
  9092   0.092%      for (int out_x = 0; out_x < output_width; ++out_x) {
                       const int in_x_origin = (out_x * stride_width) - pad_width;
 46142   0.467%        for (int out_channel = 0; out_channel < output_depth; ++out_channel) {
 52460   0.531%          auto group = out_channel / filters_per_group;
 19562   0.198%          int32_t acc = 0;
596782   6.044%          for (int filter_y = 0; filter_y < filter_height; ++filter_y) {
                           const int in_y = in_y_origin + dilation_height_factor * filter_y;
390154   3.951%            for (int filter_x = 0; filter_x < filter_width; ++filter_x) {
132192   1.339%              const int in_x = in_x_origin + dilation_width_factor * filter_x;
                             // Zero padding by omitting the areas outside the image.
                             const bool is_point_inside_image =
                                 (in_x >= 0) && (in_x < input_width) && (in_y >= 0) &&
                                 (in_y < input_height);

435865   4.414%              if (!is_point_inside_image) {
                               continue;
                             }

491681   4.980%              for (int in_channel = 0; in_channel < filter_input_depth;
188064   1.905%                   ++in_channel) {
                               int32_t input_val =
                                   input_data[Offset(input_shape, batch, in_y, in_x,
                                                     in_channel + group * filter_input_depth)];
376128   3.809%                int32_t filter_val = filter_data[Offset(
                                   filter_shape, out_channel, filter_y, filter_x, in_channel)];
790788   8.009%                acc += filter_val * (input_val + input_offset);
                             }
                           }
                         }
```

- Image convolution (>89%) has a major impact on overall performance

# Convolution accelerator structure



Pixel-by-pixel image load from memory (or get from camera)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 3 | 5 | 2 | 4 | 2 | 0 | 8 |
| 2 | 4 | 1 | 8 | 9 | 7 | 4 | 1 |
| 7 | 9 | 8 | 3 | 9 | 7 | 0 | 9 |
| 3 | 5 | 7 | 9 | $p_0$ | $p_1$ | | |
| | | | | $p_2$ | $p_3$ | 5 | 7 |
| 9 | 0 | .... | | | | | |

Weights are the same for the whole image: could be stored in dedicated registers

Result = $w_0 * p_0 + w_1 * p_1 + w_2 * p_2 + w_3 * p_3$

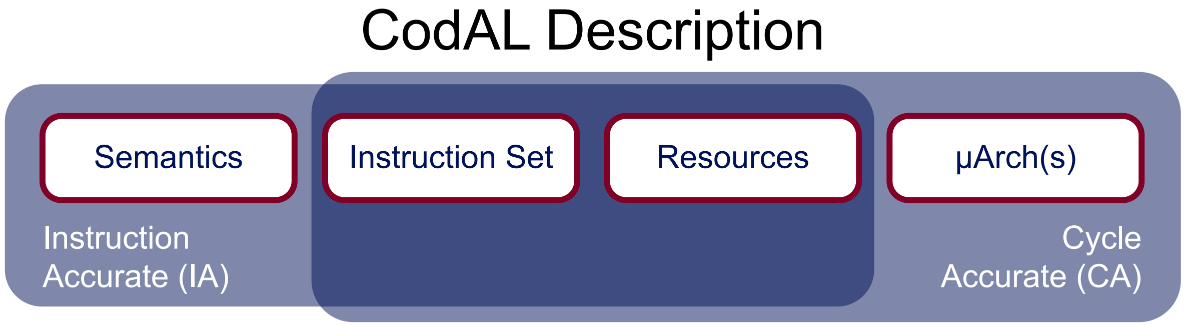
- Standard ISA, 2x2 convolution:
  - lw: 4, mul: 4, add: 3, + cycles for pipeline stall
- Convolution accelerator with FIFO:
  - Pipelined loads, 1-cycle access to all ($p_0$,$p_1$,$p_2$,$p_3$)
  - Parallel multiplications

# CodAL language

Processor description at the high abstraction level

# CodAL - processor description at the high level

- CodAL is a **C/C++ - like language** that is focused on modeling a rich set of processor capabilities
- Covers both **architecture** and **microarchitecture**
- **Instructions** are described in the form of “**elements**” that capture syntax, binary encoding and implementation
- CodAL description could be converted to RTL and used for C/C++ Compiler generation with cycle-accurate simulator and profiler



```
/*    Multiply and accumulate: semantics
          dst += src1 * src2                                        */

element i_mac {
     use reg as dst, src1, src2;
     assembly { “mac” dst “,” src1 “,” src2 };
     binary { OP_MAC dst src1 src2 0:bit[9] };
     semantics {
          rf[dst] += rf[src1] * rf[src2];
     };
};
```

# CodAL compactness

SystemVerilog

codal

```
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:**

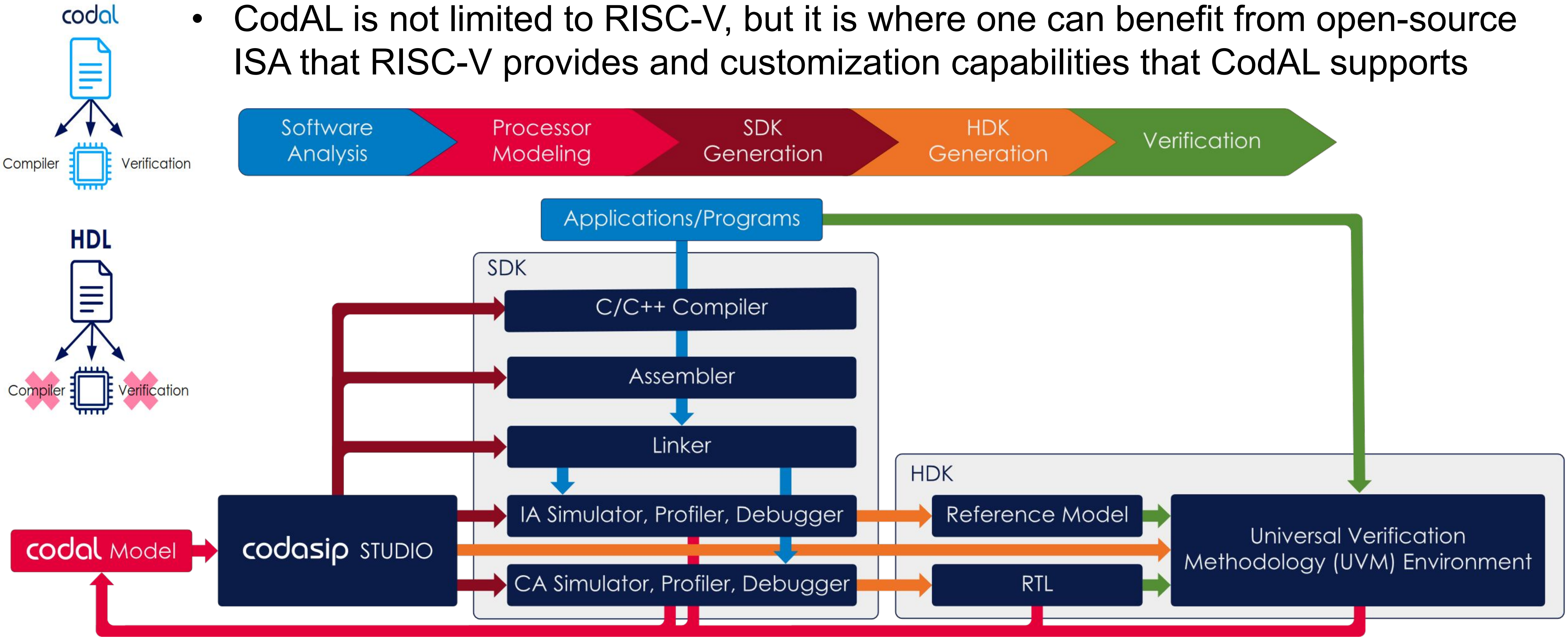
- Register files
- Memories (cache, TCM)
- On-chip debugger, trace, etc

**Many tasks are automated when using CodAL.**

- Automatic modules interconnect, decoder generation

# CodAL-based processor design flow

- CodAL is not limited to RISC-V, but it is where one can benefit from open-source ISA that RISC-V provides and customization capabilities that CodAL supports



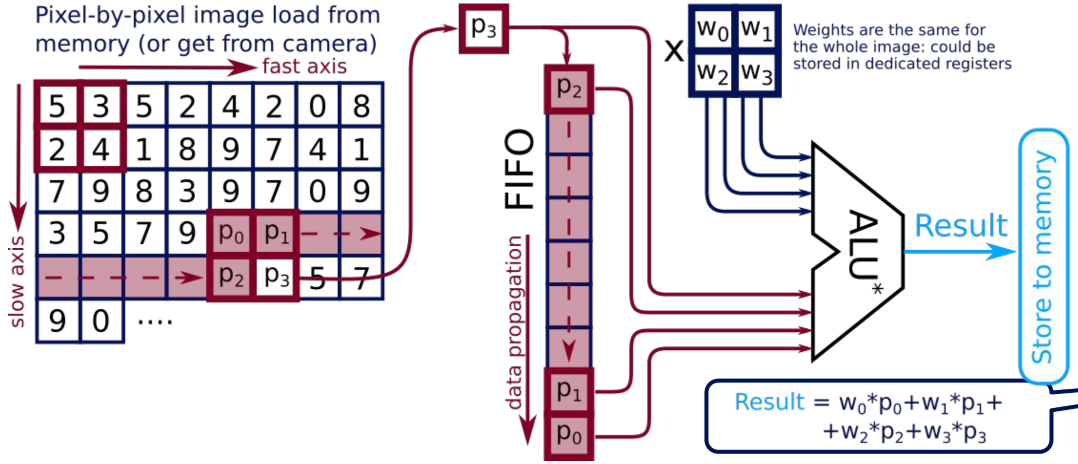
# Compact Convolution Accelerator

CodAL implementation

# CONV accelerator in <200 lines of CodAL code

**element** comprises convolution instruction assembly, binary and semantics in dedicated sections

**push_conv** instruction pushes image pixel from **src** to ***fifo[]***, calculates the convolution result and stores it to the **dst** address

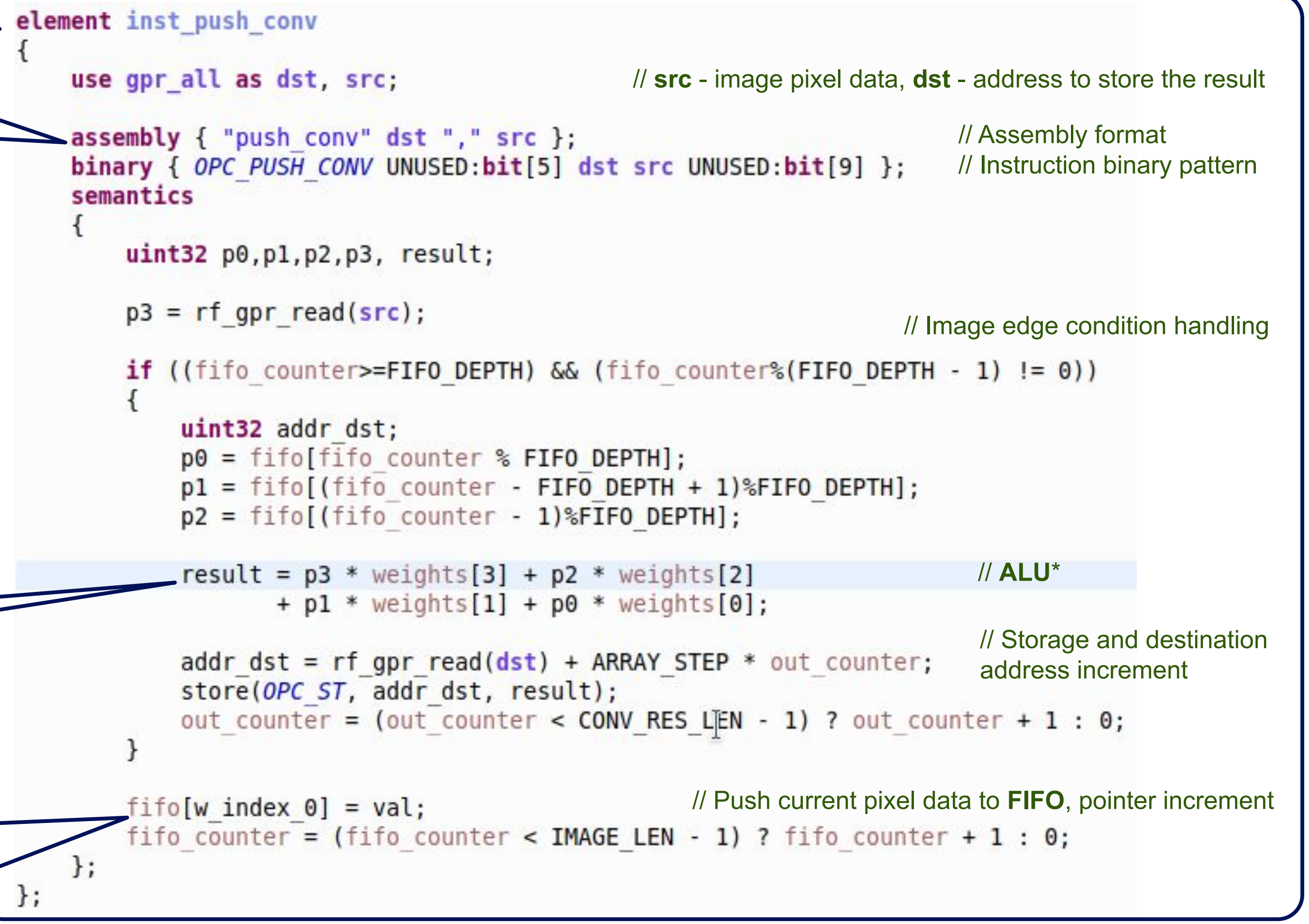

Pixel-by-pixel image load from memory (or get from camera)
fast axis
slow axis
p₃
FIFO
data propagation
p₂
p₁
p₀
x
w₀ w₁
w₂ w₃
Weights are the same for the whole image: could be stored in dedicated registers
ALU*
Result
Store to memory

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 3 | 5 | 2 | 4 | 2 | 0 | 8 |
| 2 | 4 | 1 | 8 | 9 | 7 | 4 | 1 |
| 7 | 9 | 8 | 3 | 9 | 7 | 0 | 9 |
| 3 | 5 | 7 | 9 | $p_0$ | $p_1$ | | |
| | | | | $p_2$ | $p_3$ | 5 | 7 |
| 9 | 0 | .... | | | | | |

Result = $w_0{*}p_0+w_1{*}p_1+$
$+w_2{*}p_2+w_3{*}p_3$

***fifo[]*** is a hidden register file,
**load** and **store** addresses are incremented automatically, thanks to **fifo_counter** and **out_counter** that contain the number of input image pixels loaded and output image pixels processed

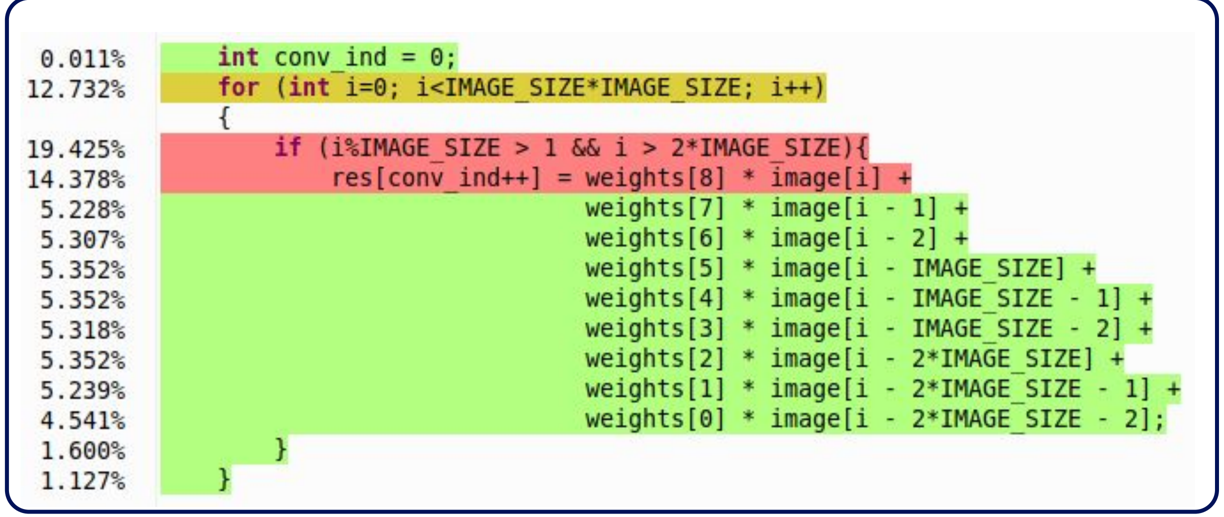
## Single instruction gets the convolution result (2x2 window)



```
element inst_push_conv
{
    use gpr_all as dst, src;                    // src - image pixel data, dst - address to store the result

    assembly { "push_conv" dst "," src };                          // Assembly format
    binary { OPC_PUSH_CONV UNUSED:bit[5] dst src UNUSED:bit[9] };  // Instruction binary pattern
    semantics
    {
        uint32 p0,p1,p2,p3, result;

        p3 = rf_gpr_read(src);
                                                                   // Image edge condition handling
        if ((fifo_counter>=FIFO_DEPTH) && (fifo_counter%(FIFO_DEPTH - 1) != 0))
        {
            uint32 addr_dst;
            p0 = fifo[fifo_counter % FIFO_DEPTH];
            p1 = fifo[(fifo_counter - FIFO_DEPTH + 1)%FIFO_DEPTH];
            p2 = fifo[(fifo_counter - 1)%FIFO_DEPTH];

            result = p3 * weights[3] + p2 * weights[2]             // ALU*
                   + p1 * weights[1] + p0 * weights[0];

            addr_dst = rf_gpr_read(dst) + ARRAY_STEP * out_counter;   // Storage and destination
            store(OPC_ST, addr_dst, result);                          address increment
            out_counter = (out_counter < CONV_RES_LEN - 1) ? out_counter + 1 : 0;
        }

        fifo[w_index_0] = val;                     // Push current pixel data to FIFO, pointer increment
        fifo_counter = (fifo_counter < IMAGE_LEN - 1) ? fifo_counter + 1 : 0;
    };
};
```

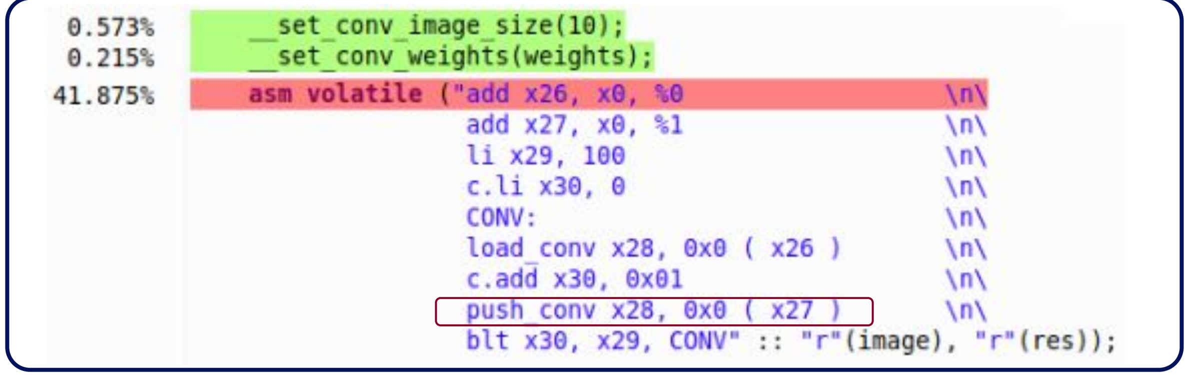
# PPA improvement for a single convolution

## Direct 3x3 convolution with standard instruction set



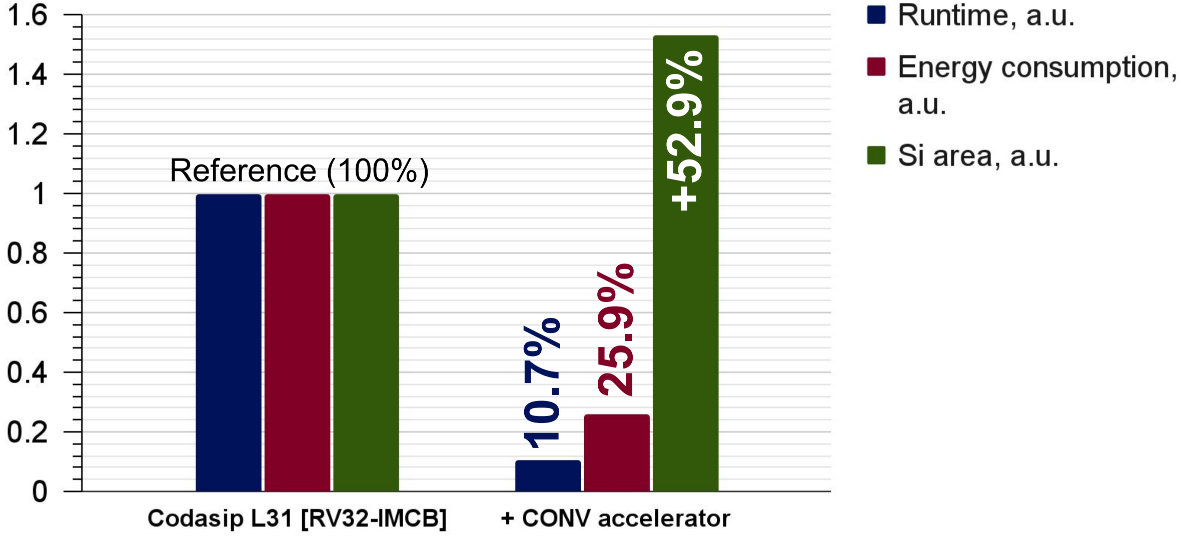
```
 0.011%    int conv_ind = 0;
12.732%    for (int i=0; i<IMAGE_SIZE*IMAGE_SIZE; i++)
           {
19.425%        if (i%IMAGE_SIZE > 1 && i > 2*IMAGE_SIZE){
14.378%            res[conv_ind++] = weights[8] * image[i] +
 5.228%                              weights[7] * image[i - 1] +
 5.307%                              weights[6] * image[i - 2] +
 5.352%                              weights[5] * image[i - IMAGE_SIZE] +
 5.352%                              weights[4] * image[i - IMAGE_SIZE - 1] +
 5.318%                              weights[3] * image[i - IMAGE_SIZE - 2] +
 5.352%                              weights[2] * image[i - 2*IMAGE_SIZE] +
 5.239%                              weights[1] * image[i - 2*IMAGE_SIZE - 1] +
 4.541%                              weights[0] * image[i - 2*IMAGE_SIZE - 2];
 1.600%        }
 1.127%    }
```

## Using custom instructions for convolution acceleration



```
 0.573%    __set_conv_image_size(10);
 0.215%    __set_conv_weights(weights);
41.875%    asm volatile ("add x26, x0, %0                  \n\
                         add x27, x0, %1                   \n\
                         li x29, 100                       \n\
                         c.li x30, 0                       \n\
                         CONV:                             \n\
                         load_conv x28, 0x0 ( x26 )        \n\
                         c.add x30, 0x01                   \n\
                         push_conv x28, 0x0 ( x27 )        \n\
                         blt x30, x29, CONV" :: "r"(image), "r"(res));
```

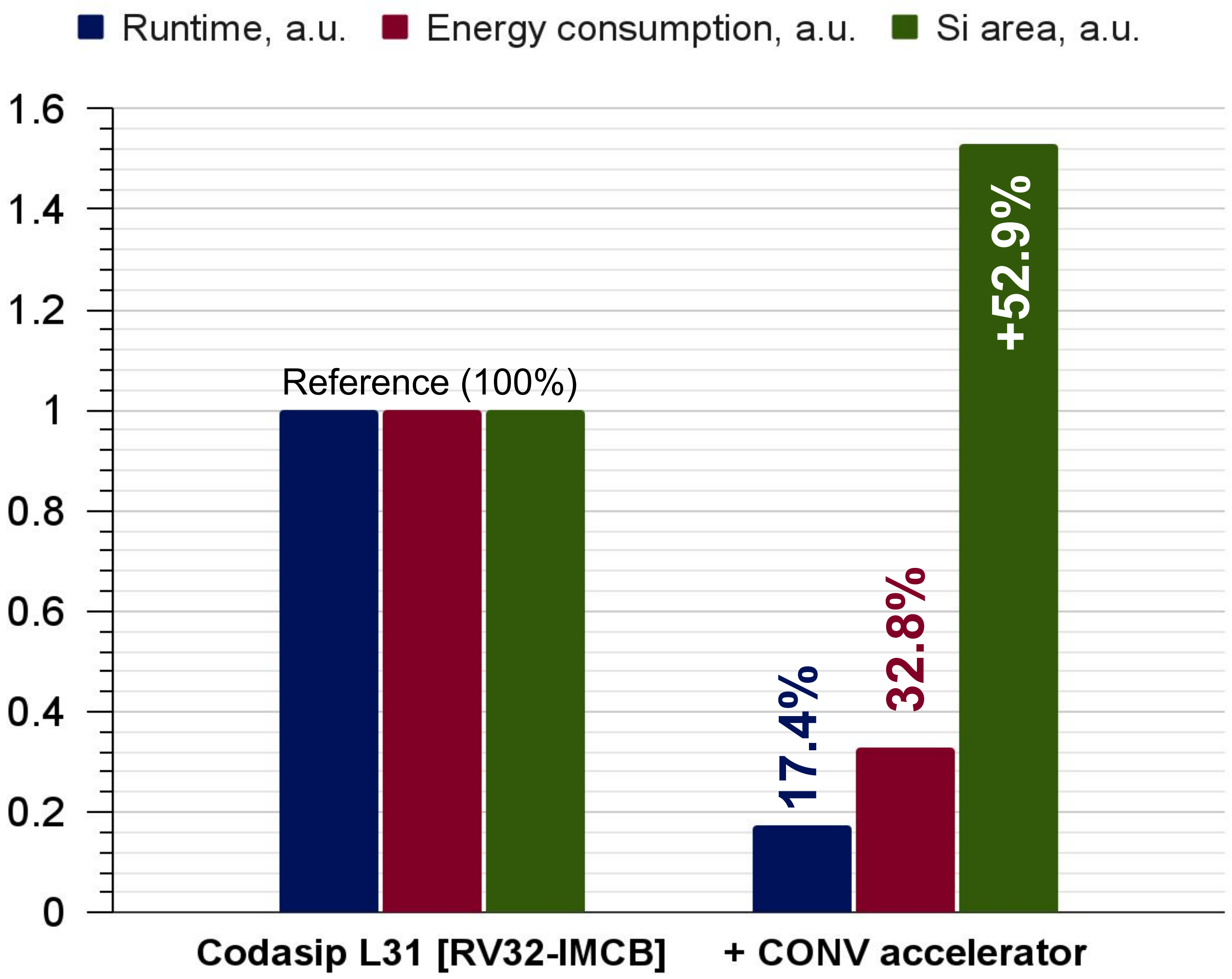
- Convolution accelerator fits 200 lines of CodAL code
- Image convolution runtime reduced down to **10.7%** of the initial value (10x10 image, 3x3 conv window)
- Average energy consumption is reduced to **25.9%**
- Si area cost **+52.9%**
- The runtime gain is expected to further scale with the image and convolution window sizes



# PPA improvement for image classification

| | Codasip L31 | Codasip L31 + CONV accelerator |
|---|---|---|
| **P**erformance, CPU cycles | 16,481,715 | 2,868,991 (17.4%) |
| **P**ower, arb. units | 100% | 32.8% |
| **A**rea, arb. units | 100% | 153% |

- Compact accelerator fitting 200 lines of code has been shown
- MNIST benchmark runtime has been reduced to **17.4%**, energy consumption - to **32.8%**
- Si area cost is **+52.9%** to that of RV32-IMCB core



# Conclusions

- The ability to accelerate image processing AI applications on embedded devices by 2D image convolution boosting has been shown
  - Runtime reduced to 17.4%
  - Power consumption lowered to 32.8%
  - 52.9% area overhead (RV32-IMCB Codasip L31 core)
- Benefits of high-level CodAL language for compact AI accelerator design:
  - Fast design space exploration
  - Less than 200 lines of convolution accelerator code
  - SDK support of custom instructions