Compact CORDIC accelerator implementation for embedded RISC-V core

Aleksei Shchekin[1], Ettore Antonino Giliberti[2]

[1]Codasip Gmbh, Germany
[2]Codasip Group, Barcelona

Abstract

Trigonometric functions are used in many embedded systems applications, such as signal and image processing, control theory, communication systems and robotics. Taking the advantage of RISC-V ISA flexibility and Codasip processor design technologies, we propose a fast and efficient implementation of the CORDIC algorithm implemented as custom instruction in an embedded RV32IMCB core in order to smoothly compute trigonometric functions. This customization improves the performance reducing the processing time by 24x, energy consumption by 13.5x at the cost of an additional 4% of silicon area. The CORDIC accelerator was implemented with 210 lines of CodAL code. Such a compact implementation alongside an automatically generated toolchain significantly shortening the time to ASIP market facilitating IP core customizations.

Introduction

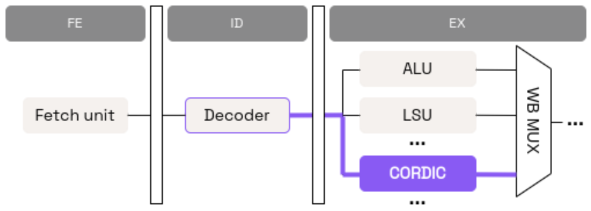
Trigonometric functions are used in embedded systems for a wide variety of signal and image processing applications, motor and power control, PWM generation [1], digital communication systems and robotics.

Embedded devices are typically resource-constrained, making it difficult to efficiently run many complex mathematical algorithms. Several methodologies exist to compute trigonometric functions, implemented either with purely software, or hardware or mixed solutions. The fastest trigonometric computation can be obtained by using look-up tables, but they require a huge memory space and corresponding area. In this scenario, the CORDIC algorithm, a hardware-efficient iterative method [2], provides a promising trade-off, since it allows computation of trigonometric primitives with no multiplications, no FPU and with relatively small additional resources. In addition, the CORDIC algorithm computes both sine and cosine values at the same time.

The CORDIC algorithm is based on an iterative input angle rotation by some specific values accompanied by (x,y) coordinate transformation to get closer and finally target the corresponding sine and cosine values. These transformations do not require multipliers by properly choosing the rotation angle values. It should be noted that the CORDIC algorithm requires an input angle and provides sine and cosine output values in fixed-point representation, so the conversion is required if one starts from floating-point representation. These conversions can be done with either hardware or software means, in our approach, they are implemented by software.

Methodology

This paper shows how the high-level synthesis tools and LLVM-based automated compiler generation simplify embedded processor customization from a software and hardware points of view. The design flow starts from the Codasip L31 RISC-V 32-bit core with 3 pipeline stages, which is extended with the CORDIC accelerator tightly coupled to a processor pipeline:



The CORDIC module resides at the EX stage and is enabled by *a single custom* instruction {“cordic” dst "," src} that takes the input angle encoded in 32-bits from the “src” register and computes the *cosine* and *sine* values. They are represented in fixed-point 16-bits format and concatenated in a single output “dst” register.

The proposed CORDIC implementation is based on a 16-bits architecture that requires 16 iterations to get the result: the input angle is rotated 16 times before getting the corresponding sine and cosine values. Each iteration takes one core clock cycle that allows one to shorten CORDIC execution path and minimize the effect on the core critical paths and its maximum working frequency. During the computation CORDIC module stalls the processor pipeline until the result is obtained after 16 cycles.

The accuracy analysis showed that this solution introduces only an additional $\mathbf{10^{-4}}$ error in the sine and cosine results, compared with the results obtained by software means with the usage of the “*math.h”* library.

The processor and CORDIC module are described in CodAL, which is a high-level architectural description language specific for processors that reduces the code volume to about 40% compared to conventional HDLs. Moreover, it simplifies customization by automated toolchain generation that supports new instructions and contains LLVM compiler, linker, debugger as well as a cycle-accurate simulator.

CodAL has a C/C++-like syntax and can handle many C constructs in a way that is familiar to software engineers, however the resultant CodAL model can be converted to RTL so that the ultimate design could be synthesized and instantiated in silicon.

The definition of custom instruction in CodAL takes the form of an "element" with the following 3 sections:

- *Assembly* section captures the instruction syntax
- *Binary* section maps the opcodes, registers and immediate values to 32 or 64 bits
- *Semantics* section describes the instruction behavior

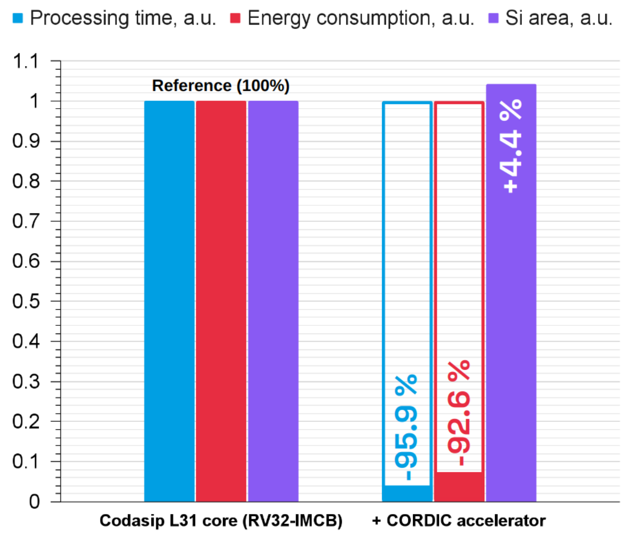
CodAL description of "cordic" instruction that enables the CORDIC module and starts the iterative "sin" and "cos" values calculation is given below:

```
***********************************
element i_cordic {
  assembly { "cordic" dst "," src };
  binary { 0:bit[12] src opc dst OPC_CORDIC};
  semantics {
    angle = rf_gpr_read(src);
    cos = 1;
    sin = 0;
    for (shift=0; shift < ITERATIONS; shift++) {
      if (angle<0) {
        cos += sin >>> shift;
        sin -= cos >>> shift;
        angle += tan[shift];
      } else {
        cos -= sin >>> shift;
        sin += cos >>> shift;
        angle -= tan[shift];
      }
    }
    rf_gpr_write(dst, (cos :: sin));
  };
};
***********************************
```

The *semantics* section comprises all actions that instruction is responsible for. First, the 32-bit "angle" value is taken from the input register "src". The iterative rotations are described within a "for-loop" where (cos, sin) are transformed using only addition and shift arithmetic operations, while the "angle" is incremented or decremented by "tan" values taken from the lookup table. Ultimately two 16-bit (cos, sin) values are written to the upper- and lower- half of the 32-bit output register.

PPA improvement

Adding CORDIC accelerator and custom instruction to control it led to a custom L31 core with better performance and total energy consumption than the standard L31. The PPA results comparison is given on diagram below, they were obtained with cycle-accurate profiling of the same trigonometric task on standard and customized L31 models:



To estimate maximum operation frequency, both the standard Codasip L31 and customized core designs were synthesized using Genus EDA tools for the TSMC 28nm tech node in multi-mode multi-corner (mmmc) flow for a range of target frequency from 50 to 650 MHz. For both designs the timing conditions were met up to 650 MHz, and the linearity of dynamic power dependence on frequency was preserved. That means customization and CORDIC accelerator did not affect maximal core frequency. The summary of the L31 core customization is given below:

Table 1: Codasip L31 with CORDIC accelerator.

TSMC 28nm	L31	CORDIC
Area, a.u.	100%	104%
Performance	1x	24.3x
Energy consumption	100%	7.4%
Lines of code	210(CodAL) vs 600(Verilog)	

The customization improves the performance lowering the processing time by 24x, energy consumption by 13.5x at the cost of additional 4% of silicon area. The compactness of CodAL language with respect to other HDLs helped to significantly reduce the design effort. CORDIC accelerator fits **210** lines of CodAL code written with the 1 person-week effort. The same module will require ~**600** lines of Verilog code. Taking a compact implementation and an automatically generated toolchain altogether allows one to significantly shorten the way to the ASIP market and simplify IP core customization.

References

[1] Christober Rayappan. *Tutorial: Software implementation of Trigonometric Functions using a CORDIC algorithm*. In: EETimes (Dec 2006).

[2] Dr. Steve Arar. *An Introduction to the CORDIC Algorithm*. In: ALL ABOUT CIRCUITS (May 2017).