CHORD: CORDIC Hardware of RISC-V Device

Ke Wu, Jiayi Chen
Department of Micor/Nano Electronics, Shanghai Jiao Tong University, Shanghai, China.
oracle0133@gmail.com, miracle3310@sjtu.edu.cn

***Abstract*—CHORD is a hardware accelerator IP for trigonometric functions. It implements the CORDIC (COordinate Rotation DIgital Computer) algorithm with a six-stage pipeline and is equipped with an AHB-Lite bus interface. This IP is designed as a co-processor of RISC-V device. We synthesis the design on T-HEAD's wujian100 platform and verify it in RTL and hardware level. Driver library in C is provided. The driver supports scalar and vector operations. The CHORD consumes 23 mW in our FPGA board and increase the hardware utilization by 3%.**

***Index Terms*—CORDIC, hardware accelerator, wujian100**

I. Introduction

CORDIC, also known as the Volder's algorithm, is an efficient algorithm to calculate trigonometric functions and their inverse functions. The implementation of the algorithm only requires addition, subtraction and bitshift units. Therefore, CORDIC based accelerators are suitable for systems where cost and power consumption are critical.

Wujian100 is a simple yet fully featured MCU base SoC developed by T-HEAD. The SoC implements a RISC-V processor based on the rv32ec ISA and fundemental peripherals, e.g., TIM (Timer), DMA (Direct Memory Access) and USI (Unified Serial Interface). Communication between modules observes the AHB protocol and a multi-master, multi-slave AHB matrix is provided.

In this paper, we present a CORDIC based hardware accelerator, which is called CHORD. CHORD implements the CORDIC algorithm with a six-stage pipeline and provides acceleration of sin, cos and arctan functions. The accelerator is designed as a co-processor of RISC-V devices, therefore, an AHB-Lite bus interface is integrated inside the system. We connected our CHORD accelerator to the processor as a peripheral via the AHB bus. We choose the wujian100 platform to realize our design and the hardware is implemented on the XC7A200T FPGA. The calculation result is transferred to a PC by UART serial port to verify our desgin.

The paper is organized as follows: section II introduces the modules of our design in detail, section III presents our verification method in HDL and hardware level, section IV is the implementation process and introduces the drivers written in C, section V discusses the results such as area consumption and calculation delay, and section VI is the conclusion of the paper.

II. System Modules

In this part we make an introduction to system modules of CHORD, including their HDL-level implementations. This system is composeds of five modules: pipeline, interface_in, interface_out, FIFO and AHB-Lite bus interface, as shown in Fig.1. The pipeline module implements CORDIC algorithm with six-stage pipeline. Interface_in and interface_out modules are used to transform data organization form between bus and pipeline. FIFO is inserted between interface_out and AHB-Lite bus interface, which serves as the buffer between CORDIC and AHB bus. AHB-Lite bus interface is the bus interface of our co-processor. The implementations of two key modules, e.g. pipeline and AHB-Lite bus interface, are described in detail below.

A. CORDIC Execution Module

To compute trigonometric values of the input degree φ, the CORDIC algorithm will start from a given vector (x_0, y_0), and then apply succesive rotations until the desired angle is reached. After n iterations the rotated vector should be $(\sin\theta_n, \cos\theta_n)$, where θ_n is approximately φ.

In each iteration i, the vector is updated through one θ_i rotation,

$$x_{i+1} = \cos\theta_i(x_i - y_i \tan\theta_i) \tag{1}$$

$$y_{i+1} = \cos\theta_i(y_i + x_i \tan\theta_i) \tag{2}$$

The insight of CORDIC is to choose those $\tan\theta_i$ in the form of 2^{-i}, so the multiplication operation can be converted in one bitshift operation, which makes an significant improvement on compute efficiency.

In order to further reduce multiplication operation, we define

$$k_i = \cos\theta_i = \cos(\arctan 2^{-i}) = \frac{1}{\sqrt{1+2^{-2i}}} \tag{3}$$

$$d_i = sign(\theta_i) \tag{4}$$

then the multipiler k_i can be avoided in each iteration, as we compute $k = \prod k_i$ as CORDIC gain and set initial vector as $(k, 0)$.

Equation 1 and 2 can thus be simplified to

$$x_{i+1} = x_i - y_i d_i 2^{-i} \tag{5}$$

$$y_{i+1} = y_i + x_i d_i 2^{-i} \tag{6}$$

and the final rotation degree is

$$\theta_n = \sum d_i \arctan 2^{-i} \tag{7}$$

The computation process of inverse trigonometric functions is similar to trigonometric functions. To compute $\arctan t$, we initialize the vector (x_0, y_0) as $(1, t)$. Obviously the angle between the initial vector and X-axis is the required result

This is the course project of MR334 supervised by Prof. Guanghui He. Special thanks to Prof. He and our TA Gang Wang.

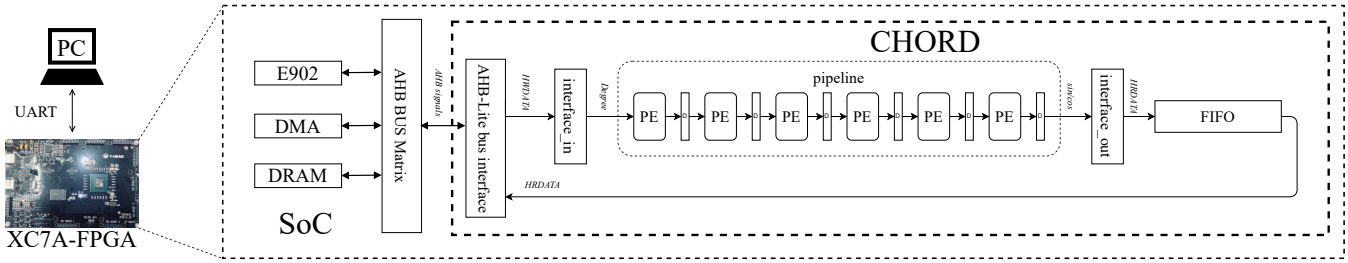

Fig. 1. Overall architecture of system

$\arctan t$ since $\tan(\arctan t) = t$. Let the initial vector be rotated to X-axis, after n iterations CORDIC will give the final rotation degree θ_n, and that is just the output.

CHORD implements CORDIC algorithm with six-stage pipeline, that is to say the iteration times $n = 6$. Declare the register variable `degree_approx_reg` to indicate the degree that has been rotated, `x_reg` and `y_reg` to indicate the vector components. In each stage i, if `degree_approx_reg[i-1]` is smaller than desired degree φ, then `degree_approx_reg[i]` adds θ_i which is stored a priori in pipeline module, meanwhile `x_reg[i]` and `y_reg[i]` are updated like

```
x_reg[i]<=x_reg[i-1]+(y_reg[i-1]>>>(i-1))
y_reg[i]<=y_reg[i-1]-(x_reg[i-1]>>>(i-1))
```

If `degree_approx_reg[i-1]` is bigger than φ, swap the above plus and minus signs.

B. AHB-Lite Bus Interface

Wujian100 supports AHB-Lite bus protocol. It is a bus interface that supports a single bus master and provides high-bandwidth operation (though CHORD requires only low-bandwidth). In our design, the AHB-Lite bus interface is slave while RISC-V kernel is master.

Since wujian100 does not support burst operation, we only implement SINGLE transfer mode, therefore `HBURST` is ignored.

Considering the width of the output from pipeline is bigger than 16bits, we set the width of the data bus to 32bit. This width is unconfigurable so `HSIZE` is ignored, too.

`HREADY` indicates if a tranfer is completed. This signal is useful when the slave want to extend the data phase, but it may also lead to endless loops in bus if `HREADY` is always set LOW. Considering the small data size we would like to just ignore `HREADY`, and thus `HREADYOUT` is not used.

We introduce finite-state machine (FSM) in our bus interface to control those AHB-Lite signals. There are four states in the interface: INIT, IDLE, READ and WRITE. State INIT occurs only once when the bus interface is initialized after `HRESETn` is set from LOW to HIGH. State IDLE means the bus interface is not selected or the master set transfer types to IDLE, that is to say state IDLE happens when `HSEL` is LOW or `HTRANS` equals to b00. State READ/WRITE means the master want to read/write from/to CHORD, which depends on `HWRITE`.

Apart from dealing with bus signals, the bus interface is connected with module interface_in and FIFO, and should transfer data with them if its state is READ or WRITE. When READ, the bus interface will enable reading from FIFO and transfer data from FIFO to `HRDATA`. When WRITE, the bus interface caches `HWDATA` in internal registers and send data to interface_in as the input data for CORDIC execution module.

III. Implementation and Drivers

In this part, we talk about the implementation of CHORD's hardware and the drivers' usages.

A. CHORD on SoC

To mount CHORD on E902 SoC, replace the instantiated module `x_main_dummy_top0` with the CORDIC coprocessor in the file `ahb_matrix_top.v`. According to `wujian100_open Userguide`, The addresss range of CHORD is `0x40010000-0x4001FFFF`.

B. Drivers

We provide five functions written in C for CHORD. Based on their processing styles, they can be catalogued into two types.

1) scalar processing: These functions are designed for direct scalar processing, i.e., send a data in `int16_t` and immediately return the corresponding function value in `int16_t`. However, using these functions for large scale data processing may be costly due to the overhead of pipeline. The scalar processing functions are:

`chord_cos`, `chord_sin` and `chord_arctan`.

2) vector processing: These functions are designed for vector processing. That is, given the pointers of the input vector and output vector's head, the data od input vector is continuously sent to CHORD and results are temporary stored in the output buffer of CHORD. The results are then sent back to output vector after all the data processing are finished. The vector can be processed is restricted by the output buffer size of CHORD, which is `CHORD_BUFFER_SIZE = 512` in our implementation.

In the cosine/sine case, the function receives a pointer of type `int16_t*` and can process at most 512 data. The result is stored in the vector of type `uint32_t*` since both cosine and sine values are returned.

In the tangent case, the result is stored in the vector of type `uint16_t*` since only degree values are required.

The vector processing functions are `chord_cos_sin_v` and `chord_arctan_v`.

IV. Verification of the System

In this part, we introduce our verification method of CHORD. Based on our design, the verification can be partitioned into two parts: verification of the functionality in HDL level and verification of the hardware implementation.

A. Functionality Verification

For a n step CORDIC algorithm with iteration steps of $\{\theta_1, \theta_2, \ldots, \theta_n\}$, the possible angles can be precisely reached are

$$\Theta = \{\sum_{i=1}^{n} d_i\theta_i | d_i \in \{-1, 1\}\}. \tag{8}$$

However, for a given input ϕ, the CORDIC cannot find its closest degree in Θ if the iteration steps set is chosen as

$$\{\arctan 2^0, \arctan 2^{-1}, \ldots, \arctan 2^{-n+1}\}. \tag{9}$$

For example, if we want to calculate the cosine and sine value of 71.56501°, the approximation sequence generated by 6-step CORDIC is $\arctan 2^0 + \arctan 2^{-1} + \arctan 2^{-2} - \arctan 2^{-3} - \arctan 2^{-4} - \arctan 2^{-5} \approx 73.11°$, but the closet degree is 70.0201°. Therefore, we set the verification criteria as

$$|f(F\phi) - f(\theta)| \leq \epsilon \tag{10}$$

in which $F\phi$ is the approximation degree by CORDIC, $\theta = \arg\min_{\theta\in\Theta}\{|\phi - \theta|\}$ and f is sin or cos. If the task of CORDIC is to figure out arc-tangent value of given input t, we have

$$|F't - \theta| \leq \epsilon \tag{11}$$

where $F't$ is the approximation degree by CORDIC. If we remember $m(\phi) = \min\{\theta | \theta \in \Theta, \theta > \phi\}$. A proper choice of ϵ should be

$$\epsilon = \max\{|f(\theta) - f(m(\theta))| \, | \theta \in \Theta\} \tag{12}$$

for $f = \sin$ or $f = \cos$, and

$$\epsilon = \max\{|\theta - m(\theta)| \, | \theta \in \Theta\}. \tag{13}$$

for arc-tangent. In the case of our six-stage CORDIC ($n = 6$), the ϵ value is shown in Table I

TABLE I
ϵ FOR COSINE, SINE AND ARC-TANGENT FUNCTIONS

f	ϵ
cos	0.0624
sin	0.0610
arctan	0.0625

We sweep the input from $-90°$ to $90°$, the comparison between the HDL simulation results and theoretical results is shown in Fig. 2 and Fig. 3.

We see that the error is bounded by 0.06, which is very close to the value of ϵ. Moreover, to fully verify our HDL design,

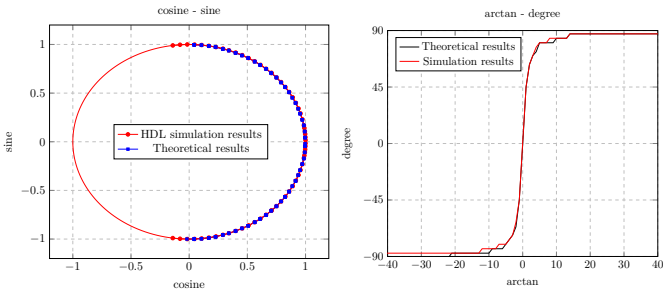

Fig. 2. CORDIC HDL simulation results v.s. theoretical results

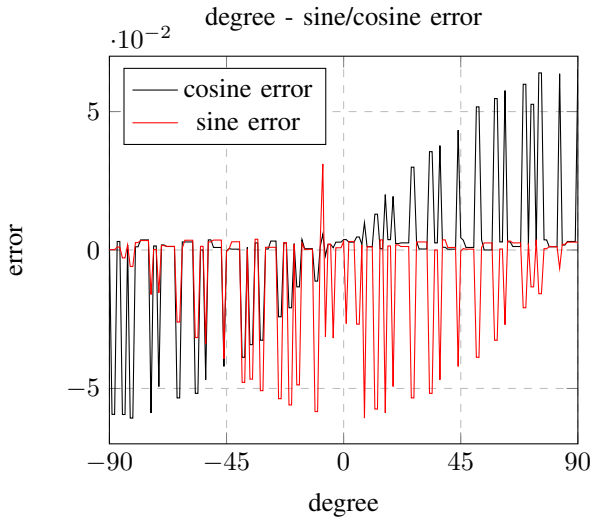

Fig. 3. Error of simulation results of functions sine and cosine

we run the simulation with random input vector with length of 512 for 1000 rounds. The result shows that our design is correct.

B. Hardware Verification

The hardware is done with XC7A200T FPGA. We transfer the calculation result back to PC by UART and redo the verification process in previous subsection. An example of UART serial port's output is shown in Fig. 4. The hardware implementation is correct according to our verification results.

V. Results

The resources utilization of CHORD is shown in Fig. 5 and Table II. We compare the implementation report of wujian100 with and without CHORD. The majorities of our hardware consumption are LUT used as logic in CORDIC and FF used as buffer in FIFO. Power consumption is shown in Fig 6. Area consumption is shown in Fig. 7.

The whole system can run in XC7A200T FPGA at the clock frequency of 20 MHz. Although a timing violation of 0.9 ns is reported, the functionality is proper since it can calculate and communicate with PC by UART effortlessly.

```
Connected.
Begin CHORD test...
Test I: non continuous sin/cos evaluation
cos, sin -90: -0.0195, -1.0000
cos, sin -89: 0.0429, -1.0000
cos, sin -88: 0.0429, -1.0000
cos, sin -87: 0.0429, -1.0000
cos, sin -86: 0.0429, -1.0000
cos, sin -85: 0.1054, -0.9960
cos, sin -84: 0.1054, -0.9960
cos, sin -83: 0.1054, -0.9960
cos, sin -82: 0.1679, -0.9882
cos, sin -81: 0.1679, -0.9882
cos, sin -80: 0.1679, -0.9882
cos, sin -79: 0.1679, -0.9882
```

Fig. 4. example of UART serial port's output

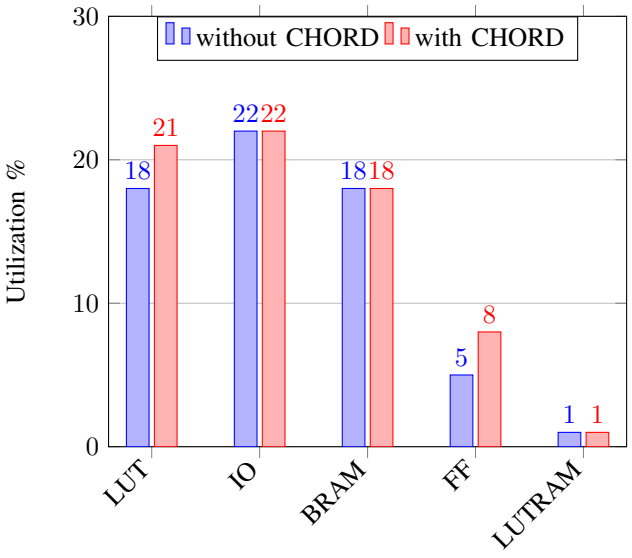

Fig. 5. Resources utilization ratio

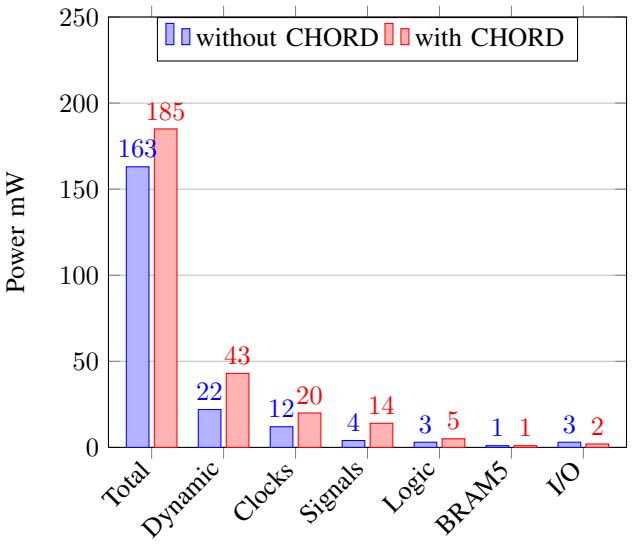

Fig. 6. Power consumption

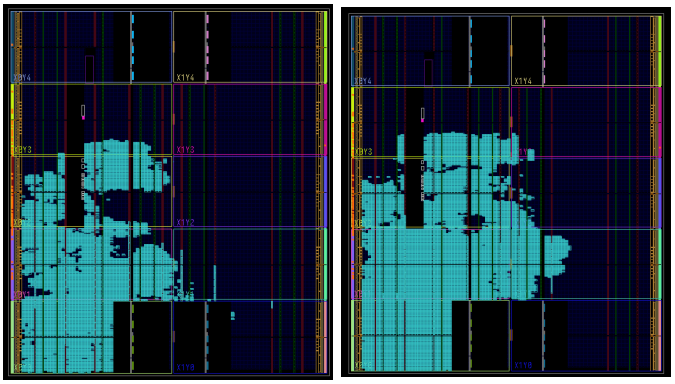

Fig. 7. left: wujian100 without CHORD, right: wujian100 with CHORD

TABLE II
RESOURCES UTILIZATION

	without CHORD	with CHOURD
LUT	24710	28912
IO	62	62
BRAM	64	64
FF	13420	22346
LUTRAM	72	74

VI. CONCLUSION

In this paper, we present a CORDIC based hardware accelerator, e.g. CHORD, and integrate it in wujian100 SoC using AHB-Lite bus interface. We also provide C functions with the help of CDK tool chain. A specific verification is performed to discover the error of the outputs from six-stage pipeline CORDIC implementation, which is within the expected range of theoretical analysis. The synthesises and implementation report shows that CHORD consumes about 23 mW in out XC7A200T FPGA platform and increases hardware utilization by 3%.

All the source code can be found in https://github.com/shrubbroom/CHORD.

REFERENCES

[1] Garcia E O, Cumplido R, Arias M. Pipelined CORDIC design on FPGA for a digital sine and cosine waves generator[C]//2006 3rd International Conference on Electrical and Electronics Engineering. IEEE, 2006: 1-4.

[2] Teja R R, Reddy P S. Sine/cosine generator using pipelined cordic processor[J]. International Journal of Engineering and Technology, 2011, 3(4): 431.

[3] AMBA® 3 AHB-Lite Protocol.

[4] wujian100_open Userguide v1.0.