[pp4fpga] CORDIC

R08943007 黃聖竣

HLS C-sim/Synthesis/Cosim (Screenshot + brief intro) :

CORDIC 演算法是一個“化繁為簡”的演算法，將許多複雜的運算轉化為一種“僅需要移位和加法”的迭代操作。應用層面最常見像是三角函數，即計算「sin，cos，sinh，cosh，tan-1」 等等

CORDIC 演算法有旋轉和向量兩個模式，分別可以在圓座標系、線性座標系和雙曲線座標系使用。

這個實驗為執行給定輸入角 θ 計算正弦和餘弦，利用 CORDIC 演算法，這些簡單的操作在硬件中使用非常有效。

以下是提供的 source code :

```
void cordic(THETA_TYPE theta, COS_SIN_TYPE &s, COS_SIN_TYPE &c)
{
  // Set the initial vector that we will rotate
  // current_cos = I; current_sin = Q
  COS_SIN_TYPE current_cos = 0.60735;
  COS_SIN_TYPE current_sin = 0.0;

  COS_SIN_TYPE factor = 1.0;
  // This loop iteratively rotates the initial vector to find the
  // sine and cosine values corresponding to the input theta angle
  L1:for (int j = 0; j < NUM_ITERATIONS; j++) {
      // Determine if we are rotating by a positive or negative angle
      int sigma = (theta < 0) ? -1 : 1;

      // Multiply previous iteration by 2^(-j)
      COS_SIN_TYPE cos_shift = current_cos * sigma * factor;
      COS_SIN_TYPE sin_shift = current_sin * sigma * factor;

      COS_SIN_TYPE cos_shift = current_cos * factor;
      COS_SIN_TYPE sin_shift = current_sin * factor;

      if (theta < 0) {
          cos_shift = -current_cos * factor;
          sin_shift = -current_sin * factor;
          theta     = theta + cordic_phase[j];
      }
      else{
          cos_shift = current_cos * factor;
          sin_shift = current_sin * factor;
          theta     = theta - cordic_phase[j];
      }


      // Perform the rotation
      current_cos = current_cos - sin_shift;
      current_sin = current_sin + cos_shift;

      // Determine the new theta
      //theta = theta - sigma * cordic_phase[j];

      factor = factor >> 1;
  }

  // Set the final sine and cosine values
  s = current_sin;  c = current_cos;
```

C-sim :

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../cordic_top.cpp in debug mode
   Compiling ../../../../cordic.cpp in debug mode
   Generating csim.exe
Total_Error_Sin=157.947048, Total_error_Cos=91.995215,
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************

```

Synthesis :

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.627 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
65	65	0.650 us	0.650 us	65	65	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	2	-	-	-
Expression	-	-	0	110	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	57	-
Register	-	-	88	-	-
Total	0	2	88	167	0
Available	280	220	106400	53200	0
Utilization (%)	0	~0	~0	~0	0

Cosim :

Cosimulation Report for 'cordic'

Result

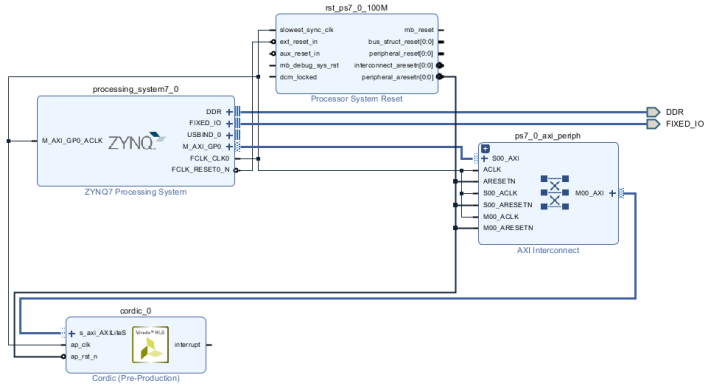
		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	65	65	65	66	66	66

Export the report(.html) using the Export Wizard

System level bring-up (Pynq or U50)

```
In [6]: from __future__ import print_function

import sys
import math

sys.path.append('/home/xilinx')
from pynq import Overlay

if __name__ == "__main__":
    print("Entry:", sys.argv[0])
    print("System argument(s):", len(sys.argv))

    print("Start of \"" + sys.argv[0] + "\"")

    ol = Overlay("/home/xilinx/IPBitFile/cordic.bit")
    regIP = ol.cordic_0

    '''for i in range(9):
        print("============================")
        for j in range(9):
            regIP.write(0x10, i + 1)
            regIP.write(0x18, j + 1)
            Res = regIP.read(0x20)
            print(str(i + 1) + " " + str(j + 1) + " = " + str(Res))'''
    terr_s = 0
    terr_c = 0
    for i in range(90):
        radian = i*math.pi/180
        s = c = 0
        regIP.write(0x10, int(radian))
        regIP.write(0x18, s)
        regIP.write(0x20, c)
        regIP.write(0x00, 0x01)

        while (regIP.read(0x00) & 0x4) == 0x0:
            continue
        err_s = math.sin(radian) - s
        err_c = math.sin(radian) - c
        terr_s += err_s
        terr_c += err_c
    print('total error sin = {}; total error cos = {}'.format(terr_s, terr_c))
    print("============================")
    print("Exit process")
```

```
Entry: /usr/lib/python3/dist-packages/ipykernel_launcher.py
System argument(s): 3
Start of "/usr/lib/python3/dist-packages/ipykernel_launcher.py"
total error sin = 56.794325064654785; total error cos = 56.794325064654785
============================
Exit process
```



Improvement - throughput, area

因為 sigma 的值為 1 或-1，用乘法器來實現的話會相對用掉不少資源，因此將它改成用 Mux 的方式來實現，另外，CORDIC 算法是一種迭代算法；因此，大多數計算都在一個 for 循環中執行，所以這邊對主要 for loop 設置 directive_pipeline II 1，由 C – Synthesis 可以看到 latency 比 baseline 少了快一半

```
L1:for (int j = 0; j < NUM_ITERATIONS; j++) {
    // Determine if we are rotating by a positive or negative angle
    //int sigma = (theta < 0) ? -1 : 1;

    // Multiply previous iteration by 2^(-j)
    //COS_SIN_TYPE cos_shift = current_cos * sigma * factor;
    //COS_SIN_TYPE sin_shift = current_sin * sigma * factor;

    COS_SIN_TYPE cos_shift = current_cos * factor;
    COS_SIN_TYPE sin_shift = current_sin * factor;

    if (theta < 0) {
        cos_shift = -current_cos * factor;
        sin_shift = -current_sin * factor;
        theta     = theta + cordic_phase[j];
    }
    else{
        cos_shift = current_cos * factor;
        sin_shift = current_sin * factor;
        theta     = theta - cordic_phase[j];
    }
```

C-sim :

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../cordic_top.cpp in debug mode
   Compiling ../../../../cordic.cpp in debug mode
   Generating csim.exe
Total_Error_Sin=157.947048, Total_error_Cos=91.995215,
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Synthesis comparisom

Performance Estimates

Timing

Clock		baseline	Improved
ap_clk	Target	10.00 ns	10.00 ns
	Estimated	8.627 ns	10.242 ns

Latency

		baseline	Improved
Latency (cycles)	min	65	34
	max	65	34
Latency (absolute)	min	0.650 us	0.348 us
	max	0.650 us	0.348 us
Interval (cycles)	min	65	34
	max	65	34

Utilization Estimates

	baseline	Improved
BRAM_18K	0	0
DSP48E	2	2
FF	110	160
LUT	263	381
URAM	0	0

Cosim :

Cosimulation Report for 'cordic'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	34	34	34	35	35	35

Export the report(.html) using the Export Wizard

Github : https://github.com/schuang23/MSOC.git