Xilinx HLS Example – atan2_CORDIC

Self-paced Report
R09943020 電子所碩一 杜承諺

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

系統介紹

- 輸入為 x,y
- 輸出 z 為(x,y)和 X 軸所夾的角度(rad)

CORDIC

- 用 atan(2^(-i))次遞迴逼近欲求出的角度，此方式在硬體實作上可以用 shift 來代替原本三角函數複雜的計算，減少面積

用四種方式做實作

- 比較直接使用 c++中 atan2 function 跟 CORDIC 的實作有何差別，並且比較四種方式跟 ground truth 的誤差
- 四種方式

```
void top_atan2(coord_t y0, coord_t x0, phase_t *zn)
{
#if defined(DB_DOUBLE_PRECISION)
    *zn = atan2( (double) y0, (double) x0);
#elif   defined(DB_SINGLE_PRECISION)
    *zn = atan2f( (float) y0, (float) x0);
#elif defined(DB_CORDIC)
    cordic_atan2(y0, x0, zn);
#else
#error <UNDEFINED ATAN2 METHOD!>
#endif
}
```

  - atan2()
    - 用 double 當作 input
  - atan2f()
    - 用 float 當作 input
  - CORDIC
    - x,y: ap_fixed<18,9>
    - z: ap_fixed<16,4>
    - 直接使用除法
  - CORDIC_bitaccurate
    - 使用 shift 代替除法

HLS CORDIC code

CORDIC (baseline)

- 用 LUT 記住需要每次旋轉的角度

```
const phase_t cord_M_PI = 3.14159265358979323846;
const phase_t cord_M_PI_2 = 1.57079632679489661923;
// phase_t cord_M_PI_4 = 0.78539816339744830961;

static const phase_t atan_2Mi[] = { 0.7854, 0.4636, 0.2450, 0.1244, 0.0624,
        0.0312, 0.0156, 0.0078, 0.0039 };
static const coord_t lut_pow2[] = {1, 2, 4, 8, 16, 32, 64, 128, 256};
```

- 遞迴去更新角度 Z，在此次例子中遞迴逼近 8 次(ROT = 8)

```
for (i = 0; i <= ROT; i++){
    if (dneg) {
        xp = x + y/lut_pow2[i];
        yp = y - x/lut_pow2[i];
        zp = z + atan_2Mi[i];
    else {
        xp = x - y/lut_pow2[i];
        yp = y + x/lut_pow2[i];
        zp = z - atan_2Mi[i];
    }
}
```

- 使用 HLS PIPELINE 來做 pipeline

```
LOOP1: for (i = 0; i <= ROT; i++)
{

#pragma HLS PIPELINE II=1
```

CORDIC (bit-accurate)

- 用 shift 代替除法

```
for (i = 0; i <= ROT; i++){
    if (dneg) {
        xp = x + y >> i;
        yp = y - x >> i;
        zp = z + atan_2Mi[i];
    else {
        xp = x - y >> i;
        yp = y + x >> i;
        zp = z - atan_2Mi[i];
    }
}
```

2. Improvement throught/area

這次優化主要放在如何使 HLS 合出我們預期的電路，共嘗試了兩種方式

- 用 shifter 來代替除法器
- 用 DSP 來取代除法器

用 shifter 來代替除法器

- 此次 CORDIC 實作中最值得一提的 improvement 就是將除法器換成 shifter，此方法在 RTL coding 是常見的優化方式，此結果證明該優化在 HLS 中也能實現
- 將除法改成 shifter 後
  - Latency: 291 → 12 (cycles) reduce 95%
  - FF: 5825 → 83 reduce 98%
  - LUT: 5071 → 844 reduce 83%

Latency

		cordic_bitaccurate	cordic
Latency (cycles)	min	2	2
	max	12	291
Latency (absolute)	min	20.000 ns	20.000 ns
	max	0.120 us	2.910 us
Interval (cycles)	min	2	2
	max	12	291

Utilization Estimates

	cordic_bitaccurate	cordic
BRAM_18K	0	0
DSP48E	0	0
FF	83	5825
LUT	844	5071
URAM	0	0

用 **DSP** 來取代除法器

- 另外我們可以觀察到和直接使用 c++提供的 library 相比，CORDIC 的實作並沒有使用到 DSP48E，因為 DSP48E 並不支援除法，導致我們不使用 shift 的方式實作，latency 甚至還比直接從 library 合成大

Timing

Clock		cordic_bitaccurate	cordic	solution1_fp32	solution1_fp64
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns	10.00 ns
	Estimated	6.935 ns	8.141 ns	8.552 ns	8.750 ns

Latency

		cordic_bitaccurate	cordic	solution1_fp32	solution1_fp64
Latency (cycles)	min	2	2	2	2
	max	12	291	147	286
Latency (absolute)	min	20.000 ns	20.000 ns	20.000 ns	20.000 ns
	max	0.120 us	2.910 us	1.470 us	2.860 us
Interval (cycles)	min	2	2	2	2
	max	12	291	147	286

- 於是將原本除法的地方改寫成乘法，可以觀察到合出來的電路有使用 DSP，並且 latency 也降到 13 cycle

```
static const coord_t lut_pow2[] = {1, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125, 0.00390625};

xp = x + y * lut_pow2[i]; //x+(y>>i);
yp = y - x * lut_pow2[i];//y-(x>>i);
```

Latency

		cordic_mul	cordic
Latency (cycles)	min	2	2
	max	13	291
Latency (absolute)	min	20.000 ns	20.000 ns
	max	0.130 us	2.910 us
Interval (cycles)	min	2	2
	max	13	291

Utilization Estimates

	cordic_mul	cordic
BRAM_18K	0	0
DSP48E	3	0
FF	116	5825
LUT	718	5071
URAM	0	0

3. Screenshot

Fp64

- **Csim**

```
total    error = 0.004645
relative error = 0.000000
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

Fp32

- **Csim**

```
total    error = 0.000000
relative error = 0.000000
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

Cordic

- **Csim**

```
total    error = 204.625488
relative error = 0.001852
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

Cordic_bitaccurate

- **Csim**

```
total    error = 204.637695
relative error = 0.001853
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

- **Cosim**

```
////////////////////////////////////////////////////////////////////////////////
$finish called at time : 13721625 ns : File "D:/NTU/MS_1/MSoC/self_paced/1_atan2_cordic/hlx_atan2
run: Time (s): cpu = 00:00:04 ; elapsed = 00:00:17 . Memory (MB): peak = 227.773 ; gain = 4.137
## quit
INFO: [Common 17-206] Exiting xsim at Thu Nov 12 17:12:58 2020...
INFO: [COSIM 212-316] Starting C post checking ...
total    error = 204.637695
relative error = 0.001853
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
Finished C/RTL cosimulation.
```

Comparison

Timing

Clock		cordic_bitaccurate	cordic	solution1_fp32	solution1_fp64
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns	10.00 ns
	Estimated	6.935 ns	8.141 ns	8.552 ns	8.750 ns

Latency

		cordic_bitaccurate	cordic	solution1_fp32	solution1_fp64
Latency (cycles)	min	2	2	2	2
	max	12	291	147	286
Latency (absolute)	min	20.000 ns	20.000 ns	20.000 ns	20.000 ns
	max	0.120 us	2.910 us	1.470 us	2.860 us
Interval (cycles)	min	2	2	2	2
	max	12	291	147	286

Utilization Estimates

	cordic_bitaccurate	cordic	solution1_fp32	solution1_fp64
BRAM_18K	0	0	4	4
DSP48E	0	0	2	3
FF	83	5825	2035	5860
LUT	844	5071	4495	10298
URAM	0	0	0	0

4. System level bring up (Pynq)

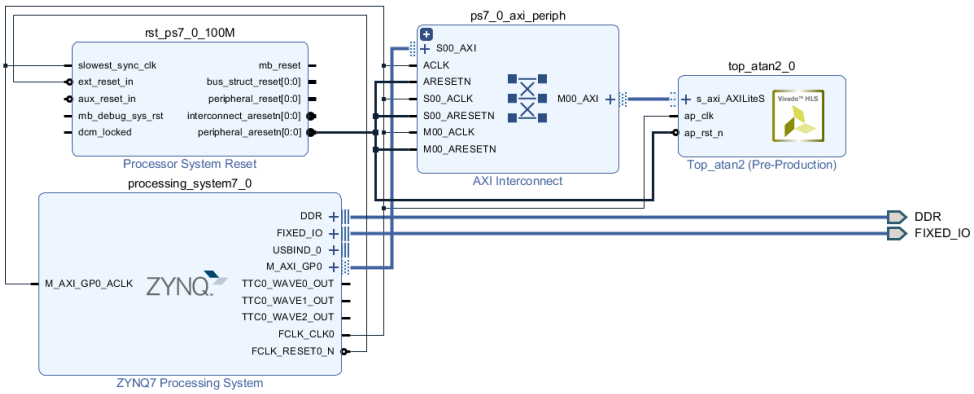
- 原先的 source code 只有提供 HLS 的部分，於是我自己實作了簡單的系統，並將 IP 放上 FPGA 作驗證
- 使用 AXILite 介面來作 I/O 的讀取

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_AXILiteS_AWVALID	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_AWREADY	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_AWADDR	in	6	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WVALID	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WREADY	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WDATA	in	32	s_axi	AXILiteS	pointer
s_axi_AXILiteS_WSTRB	in	4	s_axi	AXILiteS	pointer
s_axi_AXILiteS_ARVALID	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_ARREADY	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_ARADDR	in	6	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RVALID	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RREADY	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RDATA	out	32	s_axi	AXILiteS	pointer
s_axi_AXILiteS_RRESP	out	2	s_axi	AXILiteS	pointer
s_axi_AXILiteS_BVALID	out	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_BREADY	in	1	s_axi	AXILiteS	pointer
s_axi_AXILiteS_BRESP	out	2	s_axi	AXILiteS	pointer
ap_clk	in	1	ap_ctrl_hs	top_atan2	return value
ap_rst_n	in	1	ap_ctrl_hs	top_atan2	return value
ap_start	in	1	ap_ctrl_hs	top_atan2	return value
ap_done	out	1	ap_ctrl_hs	top_atan2	return value
ap_idle	out	1	ap_ctrl_hs	top_atan2	return value
ap_ready	out	1	ap_ctrl_hs	top_atan2	return value

- Block diagram



- PYNQ host program
  - 因為 AXILite 介面一個值是 32bit，而 IP 的 output 是 16bit，所以在讀值時需要轉換成 16bit

```
for i in range(N):
    regIP.write(0x10, int(y0[i]))
    regIP.write(0x18, int(x0[i]))
    while(regIP.read(0x24) & 0x1) == 0x0:
        continue
    Res = regIP.read(0x20)

    # convert int to fix(16,4)
    if( Res >= 32768):
        Res = (Res - 65536)
    Res = Res * (2**(-12))

    # error record
    diff = Res - float(ref_data[i])
    if diff < 0:
        diff = -diff
    total_error += diff
```

- Host program 執行結果

```
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 = 1459.3967900730468
relative error = 0.01321139537476166
============================
Exit process
```

5. Github submit

- https://github.com/MRdudu156/MSoC-2020-Fall-self-paced