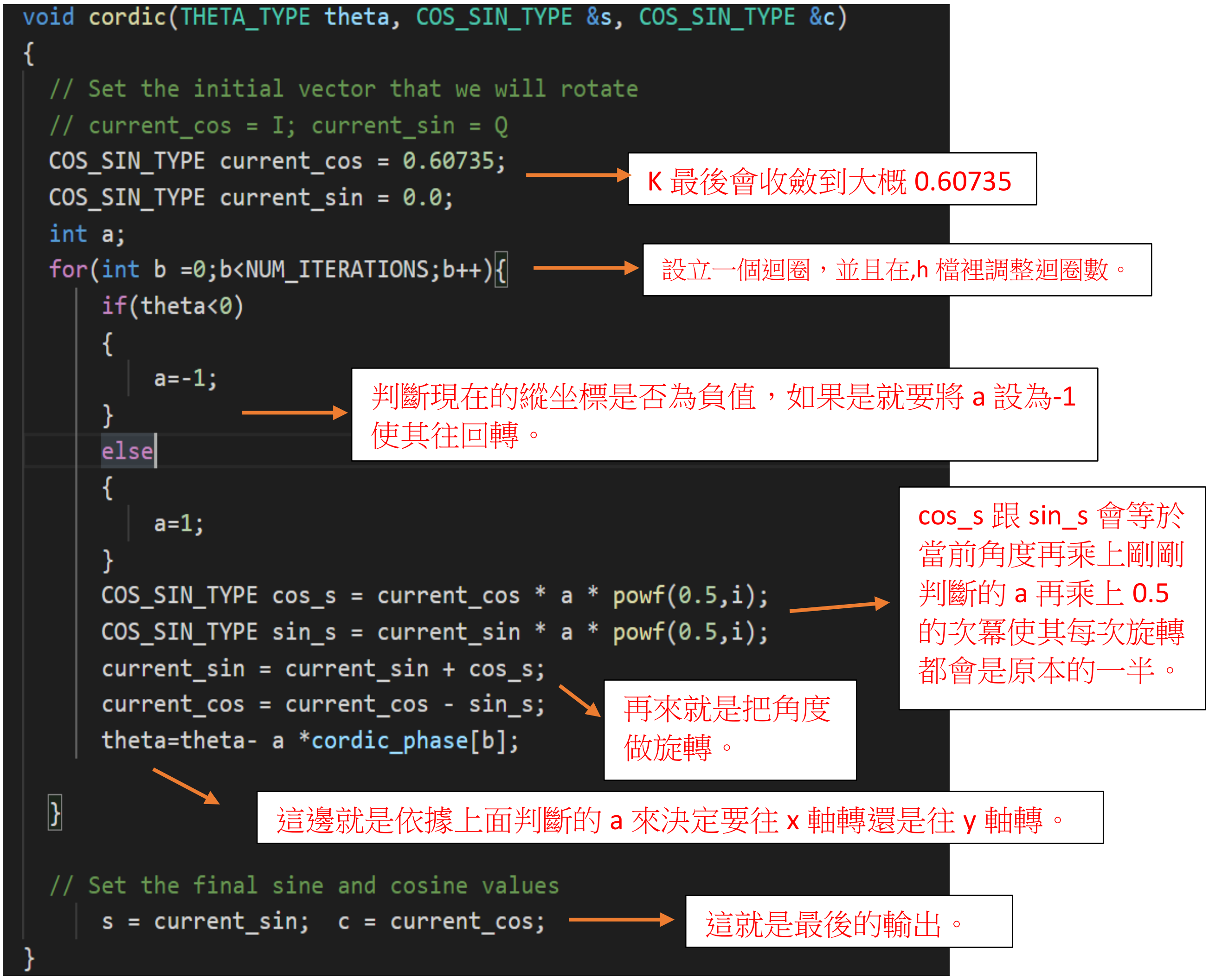
Lab1:

程式碼解說:



```
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;
  int a;
  for(int b =0;b<NUM_ITERATIONS;b++){
      if(theta<0)
      {
          a=-1;
      }
      else
      {
          a=1;
      }
      COS_SIN_TYPE cos_s = current_cos * a * powf(0.5,i);
      COS_SIN_TYPE sin_s = current_sin * a * powf(0.5,i);
      current_sin = current_sin + cos_s;
      current_cos = current_cos - sin_s;
      theta=theta- a *cordic_phase[b];

  }

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

K 最後會收斂到大概 0.60735

設立一個迴圈，並且在,h 檔裡調整迴圈數。

判斷現在的縱坐標是否為負值，如果是就要將 a 設為-1 使其往回轉。

cos_s 跟 sin_s 會等於當前角度再乘上剛剛判斷的 a 再乘上 0.5 的次冪使其每次旋轉都會是原本的一半。

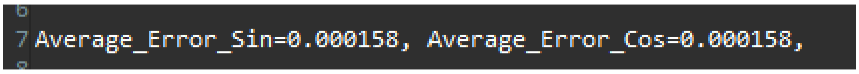
再來就是把角度做旋轉。

這邊就是依據上面判斷的 a 來決定要往 x 軸轉還是往 y 軸轉。

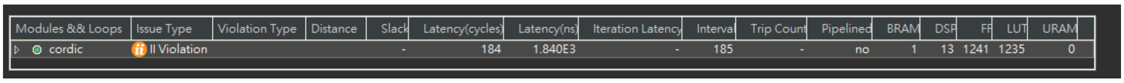
這就是最後的輸出。

測試結果:

NUM_ITERATIONS=20:誤差蠻小的，但所需的 cycles 較多

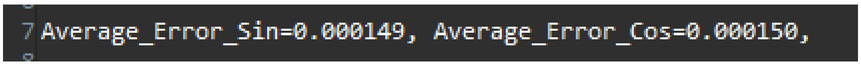


```
Average_Error_Sin=0.000158, Average_Error_Cos=0.000158,
```

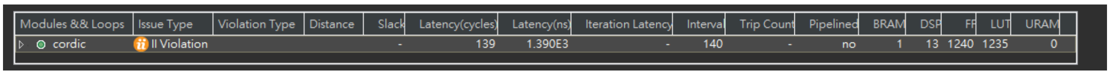


Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordic	II Violation			-	184	1.840E3	-	185	-	no	1	13	1241	1235	0

NUM_ITERATIONS=15:這是我認為的最佳方案，我試出來誤差最低且所需 cycles 較少的數字。

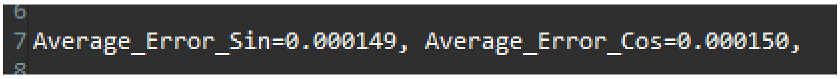


```
Average_Error_Sin=0.000149, Average_Error_Cos=0.000150,
```

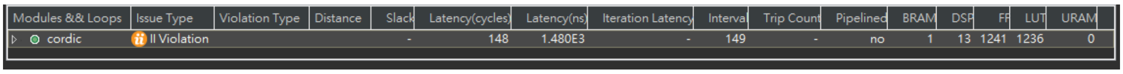


Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordic	II Violation			-	139	1.390E3	-	140	-	no	1	13	1240	1235	0

NUM_ITERATIONS=16:這個其實跟上面一個相差非常接近，但 cycles 比較多，最後就沒有選擇。



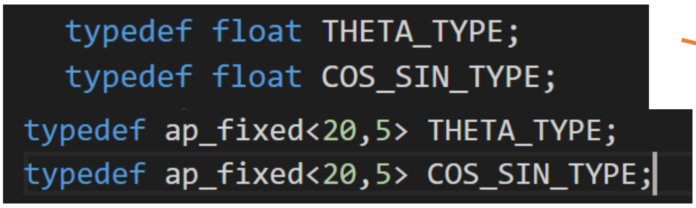
```
Average_Error_Sin=0.000149, Average_Error_Cos=0.000150,
```



Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordic	II Violation			-	148	1.480E3	-	149	-	no	1	13	1241	1236	0

Lab2:

程式碼解說:

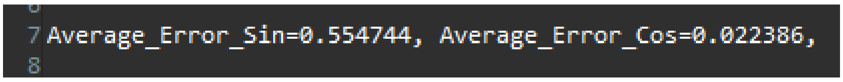


```
    typedef float THETA_TYPE;
    typedef float COS_SIN_TYPE;
typedef ap_fixed<20,5> THETA_TYPE;
typedef ap_fixed<20,5> COS_SIN_TYPE;
```

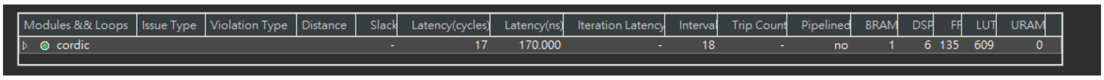
原來在.h 檔所宣告的 theta_type 及 cos_sin_type 將其改成用下面的方式宣告。其餘主程式碼不變，並且透過調整上面參數來達到最佳的結果。

測試結果(NUM_ITERATIONS 固定=15):

Ap_fixed<32,1>:誤差太大不予採納。

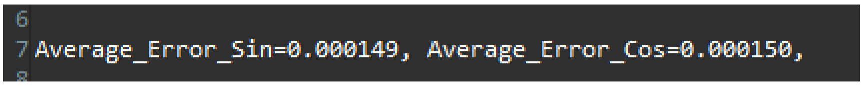


```
Average_Error_Sin=0.554744, Average_Error_Cos=0.022386,
```



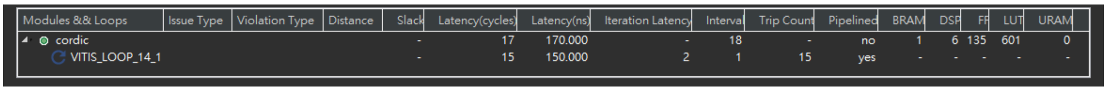
Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordic				-	17	170.000	-	18	-	no	1	6	135	609	0

Ap_fixed<32,5>:誤差已經穩定，所以後面放 5 是 OK 的，接下來就是調整前面



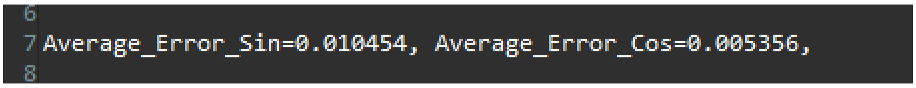
```
Average_Error_Sin=0.000149, Average_Error_Cos=0.000150,
```

。

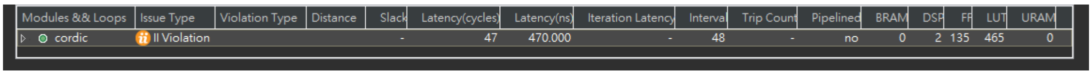


Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordic				-	17	170.000	-	18	-	no	1	6	135	601	0
VITIS_LOOP_14_1				-	15	150.000	2	1	15	yes	-	-	-	-	-

Ap_fixed<16,5>:誤差值整個超過 1%，不予採納。

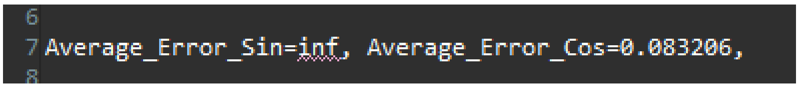


```
Average_Error_Sin=0.010454, Average_Error_Cos=0.005356,
```

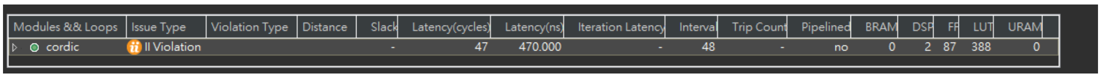


Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordic	II Violation			-	47	470.000	-	48	-	no	0	2	135	465	0

Ap_fixed<10,5>:誤差值也是差太多，不予採納。

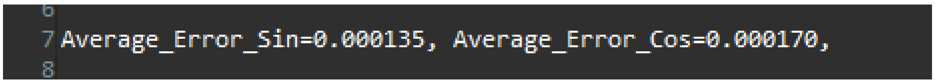


```
Average_Error_Sin=inf, Average_Error_Cos=0.083206,
```

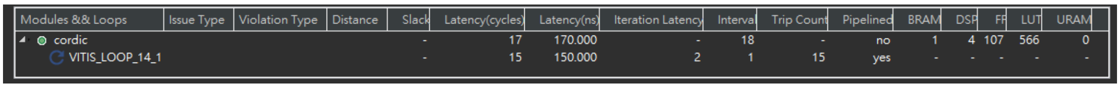


Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
▷ ● cordic	II Violation			-	47	470.000	-	48	-	no	0	2	87	388	0

Ap_fixed<25,5>:誤差穩定，但 LUT 跟 FF 可能還有改善空間。

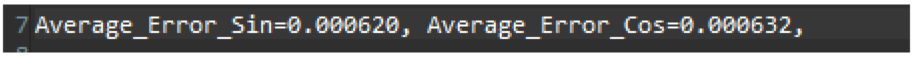


```
Average_Error_Sin=0.000135, Average_Error_Cos=0.000170,
```

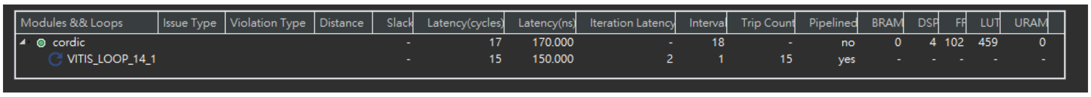


Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
◢ ● cordic				-	17	170.000	-	18	-	no	1	4	107	566	0
VITIS_LOOP_14_1				-	15	150.000	2	1	15	yes	-	-	-	-	-

Ap_fixed<20,5>:誤差值雖然高了點，但我認為可以容許，且這個參數出來的cycles 跟上一個一樣，但LUT 跟 FF 又比上一個來的少，所以最後用這個。



```
Average_Error_Sin=0.000620, Average_Error_Cos=0.000632,
```

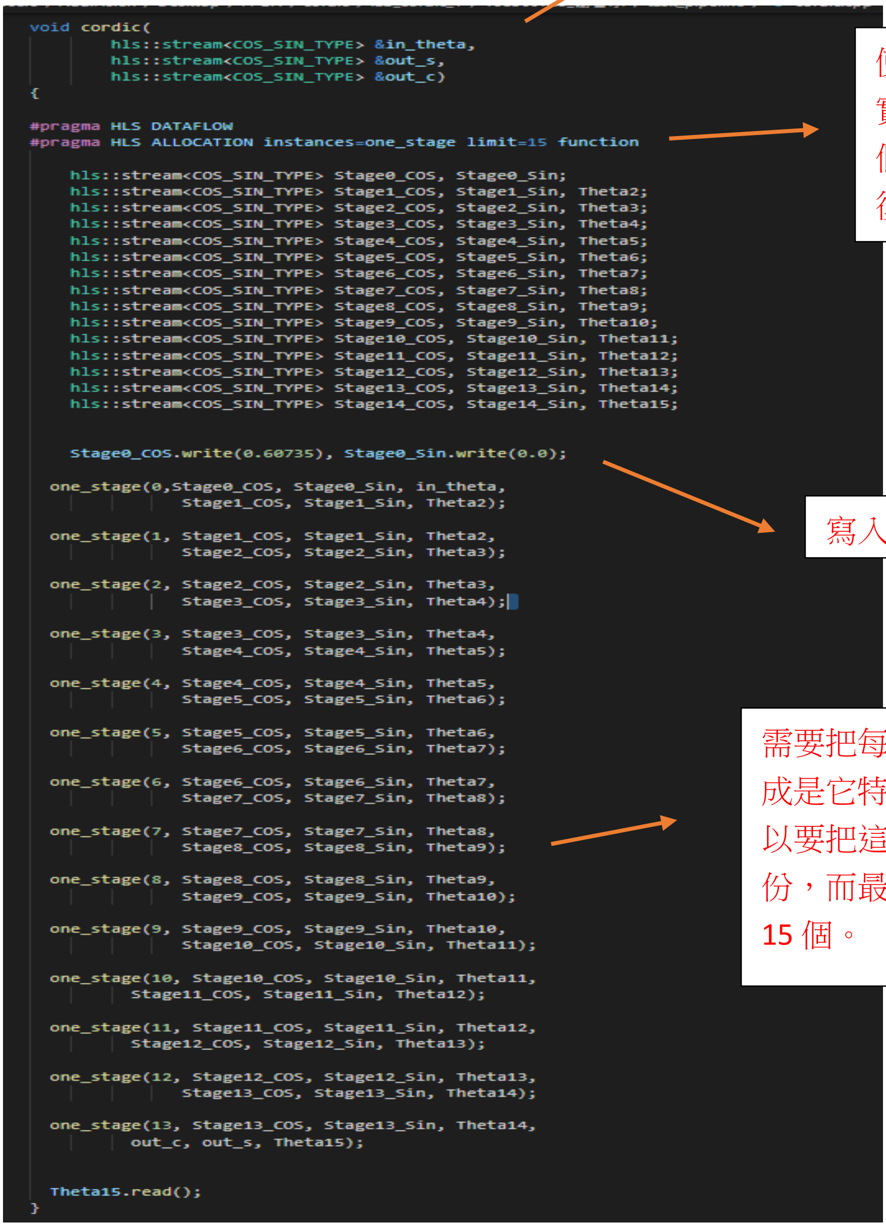


Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
◢ ● cordic				-	17	170.000	-	18	-	no	0	4	102	459	0
VITIS_LOOP_14_1				-	15	150.000	2	1	15	yes	-	-	-	-	-

Lab3:

程式碼解說:

先定義 interface，輸入為 theta，輸出為 sin 值跟 cos 值。

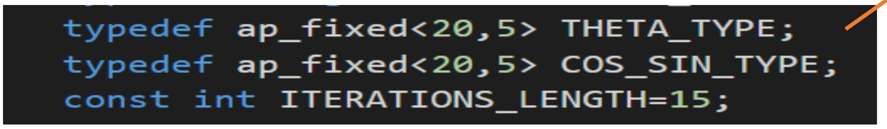


```
void cordic(
        hls::stream<COS_SIN_TYPE> &in_theta,
        hls::stream<COS_SIN_TYPE> &out_s,
        hls::stream<COS_SIN_TYPE> &out_c)
{

#pragma HLS DATAFLOW
#pragma HLS ALLOCATION instances=one_stage limit=15 function

    hls::stream<COS_SIN_TYPE> Stage0_COS, Stage0_Sin;
    hls::stream<COS_SIN_TYPE> Stage1_COS, Stage1_Sin, Theta2;
    hls::stream<COS_SIN_TYPE> Stage2_COS, Stage2_Sin, Theta3;
    hls::stream<COS_SIN_TYPE> Stage3_COS, Stage3_Sin, Theta4;
    hls::stream<COS_SIN_TYPE> Stage4_COS, Stage4_Sin, Theta5;
    hls::stream<COS_SIN_TYPE> Stage5_COS, Stage5_Sin, Theta6;
    hls::stream<COS_SIN_TYPE> Stage6_COS, Stage6_Sin, Theta7;
    hls::stream<COS_SIN_TYPE> Stage7_COS, Stage7_Sin, Theta8;
    hls::stream<COS_SIN_TYPE> Stage8_COS, Stage8_Sin, Theta9;
    hls::stream<COS_SIN_TYPE> Stage9_COS, Stage9_Sin, Theta10;
    hls::stream<COS_SIN_TYPE> Stage10_COS, Stage10_Sin, Theta11;
    hls::stream<COS_SIN_TYPE> Stage11_COS, Stage11_Sin, Theta12;
    hls::stream<COS_SIN_TYPE> Stage12_COS, Stage12_Sin, Theta13;
    hls::stream<COS_SIN_TYPE> Stage13_COS, Stage13_Sin, Theta14;
    hls::stream<COS_SIN_TYPE> Stage14_COS, Stage14_Sin, Theta15;


    stage0_COS.write(0.60735), stage0_Sin.write(0.0);

  one_stage(0,Stage0_COS, Stage0_Sin, in_theta,
                Stage1_COS, Stage1_Sin, Theta2);

  one_stage(1, Stage1_COS, Stage1_Sin, Theta2,
                Stage2_COS, Stage2_Sin, Theta3);

  one_stage(2, Stage2_COS, Stage2_Sin, Theta3,
                Stage3_COS, Stage3_Sin, Theta4);

  one_stage(3, Stage3_COS, Stage3_Sin, Theta4,
                Stage4_COS, Stage4_Sin, Theta5);

  one_stage(4, Stage4_COS, Stage4_Sin, Theta5,
                Stage5_COS, Stage5_Sin, Theta6);

  one_stage(5, Stage5_COS, Stage5_Sin, Theta6,
                Stage6_COS, Stage6_Sin, Theta7);

  one_stage(6, Stage6_COS, Stage6_Sin, Theta7,
                Stage7_COS, Stage7_Sin, Theta8);

  one_stage(7, Stage7_COS, Stage7_Sin, Theta8,
                Stage8_COS, Stage8_Sin, Theta9);

  one_stage(8, Stage8_COS, Stage8_Sin, Theta9,
                Stage9_COS, Stage9_Sin, Theta10);

  one_stage(9, Stage9_COS, Stage9_Sin, Theta10,
                Stage10_COS, Stage10_Sin, Theta11);

  one_stage(10, Stage10_COS, Stage10_Sin, Theta11,
            Stage11_COS, Stage11_Sin, Theta12);

  one_stage(11, Stage11_COS, Stage11_Sin, Theta12,
            Stage12_COS, Stage12_Sin, Theta13);

  one_stage(12, Stage12_COS, Stage12_Sin, Theta13,
                Stage13_COS, Stage13_Sin, Theta14);

  one_stage(13, Stage13_COS, Stage13_Sin, Theta14,
            out_c, out_s, Theta15);


  Theta15.read();
}
```

使用 dataflow 的 pragma 來實作 pipeline，然後因為我們知道 iteration 進行 15 次後會有比較好的結果。

寫入初始值。

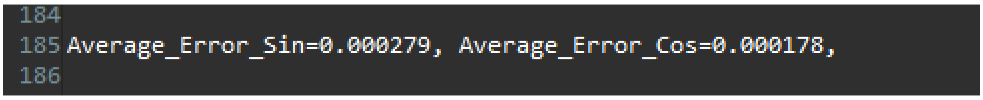
需要把每個 iteration 都實作成是它特定的 hardware，所以要把這個 function 複製 14 份，而最後的 output 是在第 15 個。



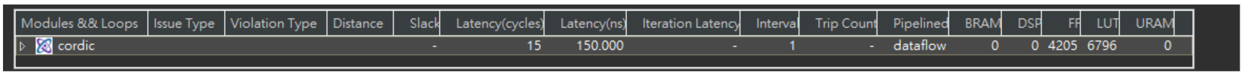
```
typedef ap_fixed<20,5> THETA_TYPE;
typedef ap_fixed<20,5> COS_SIN_TYPE;
const int ITERATIONS_LENGTH=15;
```

這是最後調整完的參數。

測試結果:



```
Average_Error_Sin=0.000279, Average_Error_Cos=0.000178,
```



Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
cordic				-	15	150.000	-	1	-	dataflow	0	0	4205	6796	0