UCSD 237C: Project 2 CORDIC

Steven Daniels

sdaniels@ucsd.edu

Student ID# A53328625

November 5, 2023

1. Introduction

This report presents a hardware design for a phase detector used to find the magnitude and phase of the output of a complex matched filter. The purpose of this report is to demonstrate the hardware implementation and its effect on resource usage and performance. One design explored:

1. Phase Detector

2. Phase Detector Design

2.1. Baseline

2.1.1. Throughput Max: **713 kHz**

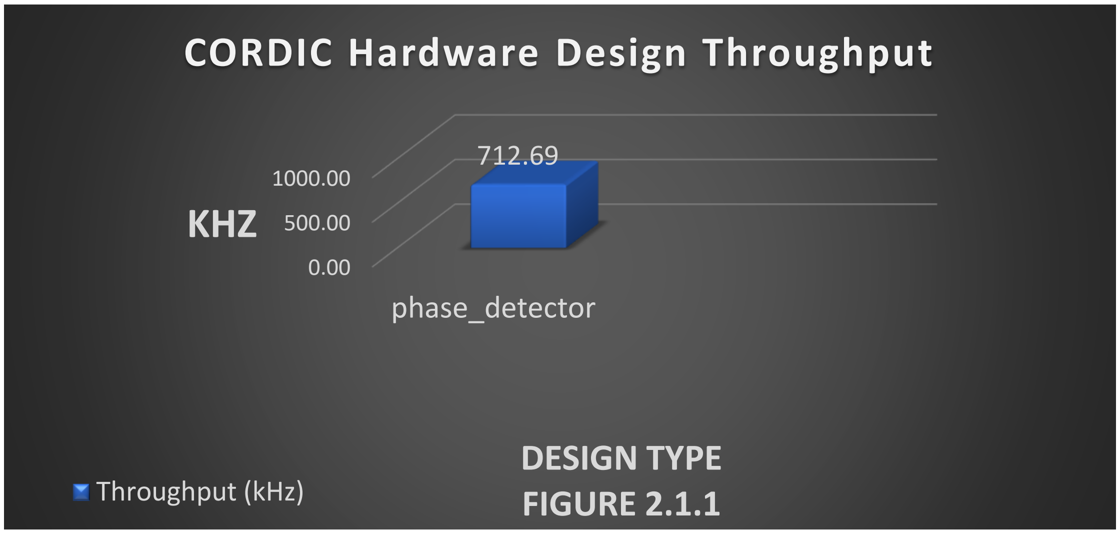


FIGURE 2.1.1

2.1.2. Implementation

```
void firI1 (
  data_t *y,
  data_t x
  ) {

    coef_t c[N] = {1,     -1,     1,     -1,     -1,     -1,     1,     1,     -1,     -
1,     -1,     1,     1,     -1,     1,     -1,     -1,     -1,     -1,     1,     1,
1,     1,     1,     -1,     -1,     1,     1,     1,     -1,     -1,     -1};
#pragma HLS bind_storage variable=c type=RAM_2P
#pragma HLS ARRAY_PARTITION variable=c type=complete
```

```
    // Write your code here
    static data_t shift_reg[N];
#pragma HLS bind_storage variable=shift_reg type=RAM_2P
#pragma HLS ARRAY_PARTITION variable=shift_reg type=cyclic factor=16

    acc_t acc;
    int i;
    acc = 0;

    TDL:
    for (i = N - 1; i > 0; i--) {
#pragma HLS PIPELINE on
#pragma HLS unroll factor=16
    shift_reg[i] = shift_reg[i - 1];
    }
    shift_reg[0] = x;
    acc = 0;

    MAC:
    for (i = N - 1; i >= 0; i--) {
#pragma HLS unroll factor=32
    acc += shift_reg[i] * c[i];
    }

    *y = acc;

}


void firI2 (
  data_t *y,
  data_t x
  ) {

    coef_t c[N] = {1,    -1,    1,    -1,    -1,    -1,    1,    1,    -1,    -
1,    -1,    1,    1,    -1,    1,    -1,    -1,    -1,    -1,    1,    1,
1,    1,    1,    -1,    -1,    1,    1,    1,    -1,    -1,    -1};
#pragma HLS bind_storage variable=c type=RAM_2P
#pragma HLS ARRAY_PARTITION variable=c type=complete

    // Write your code here
    static data_t shift_reg[N];
#pragma HLS bind_storage variable=shift_reg type=RAM_2P
#pragma HLS ARRAY_PARTITION variable=shift_reg type=cyclic factor=16

    acc_t acc;
    int i;
    acc = 0;

    TDL:
    for (i = N - 1; i > 0; i--) {
#pragma HLS PIPELINE on
#pragma HLS unroll factor=16
    shift_reg[i] = shift_reg[i - 1];
    }
    shift_reg[0] = x;
    acc = 0;

    MAC:
    for (i = N - 1; i >= 0; i--) {
#pragma HLS unroll factor=32
    acc += shift_reg[i] * c[i];
    }

    *y = acc;

}




void firQ1 (
  data_t *y,
  data_t x
  ) {

    coef_t c[N] = {-1,    -1,    1,    -1,    1,    -1,    1,    -1,    -1,
-1,    -1,    1,    -1,    1,    -1,    1,    1,    -1,    1,    -1,    -1,
1,    -1,    1,    1,    1,    1,    -1,    1,    -1,    1,    1};

#pragma HLS bind_storage variable=c type=RAM_2P
#pragma HLS ARRAY_PARTITION variable=c type=complete

    // Write your code here
    static data_t shift_reg[N];
#pragma HLS bind_storage variable=shift_reg type=RAM_2P
#pragma HLS ARRAY_PARTITION variable=shift_reg type=cyclic factor=16

    acc_t acc;
    int i;
    acc = 0;

    TDL:
    for (i = N - 1; i > 0; i--) {
#pragma HLS PIPELINE on
#pragma HLS unroll factor=16
    shift_reg[i] = shift_reg[i - 1];
    }
    shift_reg[0] = x;
    acc = 0;

    MAC:
    for (i = N - 1; i >= 0; i--) {
#pragma HLS unroll factor=32
    acc += shift_reg[i] * c[i];
    }

    *y = acc;

}

void firQ2 (
  data_t *y,
  data_t x
  ) {

    coef_t c[N] = {-1,    -1,    1,    -1,    1,    -1,    1,    -1,    -1,
-1,    -1,    1,    -1,    1,    -1,    1,    1,    -1,    1,    -1,    -1,
1,    -1,    1,    1,    1,    1,    -1,    1,    -1,    1,    1};
#pragma HLS bind_storage variable=c type=RAM_2P
#pragma HLS ARRAY_PARTITION variable=c type=complete

    // Write your code here
    static data_t shift_reg[N];
#pragma HLS bind_storage variable=shift_reg type=RAM_2P
```

```
#pragma HLS ARRAY_PARTITION variable=shift_reg type=cyclic factor=16

    acc_t acc;
    int i;
    acc = 0;

    TDL:
    for (i = N - 1; i > 0; i--) {
#pragma HLS PIPELINE on
#pragma HLS unroll factor=16
    shift_reg[i] = shift_reg[i - 1];
    }
    shift_reg[0] = x;
    acc = 0;

    MAC:
    for (i = N - 1; i >= 0; i--) {
#pragma HLS unroll factor=32
    acc += shift_reg[i] * c[i];
    }

    *y = acc;

}


void fir (
  data_t I,
  data_t Q,

  data_t *X,
  data_t *Y
  ) {

    data_t iValues[2];
    data_t qValues[2];

    firI1(&iValues[0],I); // i1in
    firQ1(&qValues[0],Q); // q1in

    firI2(&qValues[1],Q); // q2in
    firQ2(&iValues[1],I); // i1in

    *X = iValues[0] + qValues[0];
    *Y = qValues[1] - iValues[1];

}
```

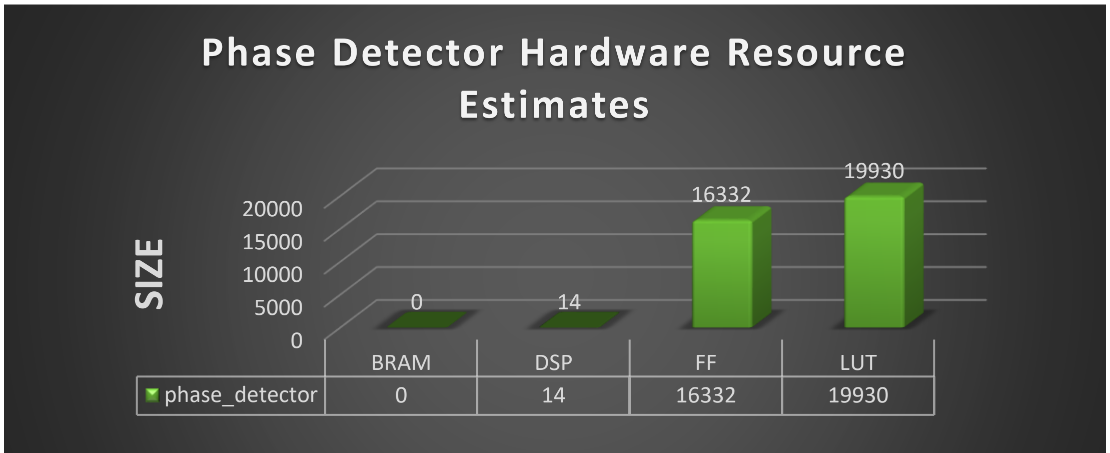
```
void cordiccart2pol(data_t x, data_t y, data_t * r,  data_t * theta)
{
    if (x == 1 && y == 0)
    {
        *r=1;
        *theta=0;
        return;
    }
    data_t cur_x = 0;
    data_t cur_y = 0;
    // Use the sign bit of x and y to determine which quadrant the vector is in
    unsigned short x_sign_bit = signbit(x);
    unsigned short y_sign_bit = signbit(y);
    int to_quad = 0;
    if(x_sign_bit == 1.0 && y_sign_bit == 0.0)
    {
        cur_x =  y;
        cur_y = -1.0*x;
        *theta = angles[0]*-2;
    }
    else if(x_sign_bit == 1.0 && y_sign_bit == 1.0)
    {
        cur_x = -1.0*y;
        cur_y = x;
        *theta = angles[0]*2;
    }
    else if(x_sign_bit == 0.0 && y_sign_bit == 1.0)
    {
        cur_x = -1.0*y;
        cur_y = x;
        *theta = angles[0]*2;
    }
    else
    {
        cur_x = y;
        cur_y = x*-1.0;
        *theta = angles[0]*-2.0;

    }

    data_t sigma = 0;
    data_t tempx = 0;
    data_t tempy = 0;
    for (int i = 0; i < NUM_ITER; i++)
    {
#pragma HLS PIPELINE off
        sigma = (cur_y > 0) ? -1 : 1;

        // perform rotation/transformation
        tempx = cur_x;
        tempy = cur_y;
        cur_x = tempx - (sigma * Kvalues[i] * tempy);
        cur_y = tempy + (sigma * Kvalues[i] * tempx);
        *theta = *theta + sigma * angles[i];
    }


    // Set the final radians and phase
    *r = cur_x*cordic_gain[NUM_ITER-1]; *theta = -1*(*theta);

}
```

```
void phasedetector (
  data_t *I,
  data_t *Q,

  data_t *R,
  data_t *Theta,

  int length
  ){
  int i;
  const int SAMPLES = 1024;
  data_t coor_X[SAMPLES];
  data_t coor_Y[SAMPLES];

    for (i = 0; i < length ;i++)
    {

        fir(I[i],Q[i],&coor_X[i],&coor_Y[i]);
        cordiccart2pol(coor_X[i],coor_Y[i],&R[i],&Theta[i]);
    }

}
```

2.1.3. Optimizations

None

2.1.4. Resources



	BRAM	DSP	FF	LUT
🟩 phase_detector	0	14	16332	19930

RESOURCE TYPE
FIGURE 2.1.4

2.1.5. Analysis

This section presents an implementation of a phase detector. It consumes a heavy number of resources and doesn't provide a high rate of throughput. A design like this would put limitations on high performing systems with moderate power requirements.

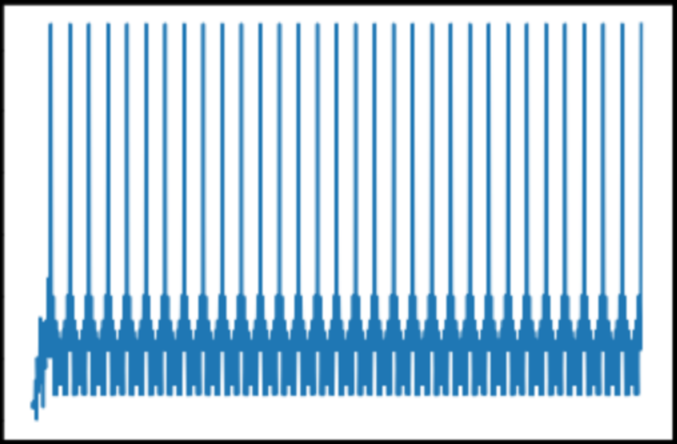
3. Phase Detector Questions

3.1. Question 1 – What is the throughput of your Phase Detector? How does that relate to the individual components (FIR, CORDIC, etc.)? How can you make it better?

- This Phase Detector design has a throughput of 713kHz. The CORDIC block consumed the highest number of cycles when compared to the fir operations. This design could be improved with function level pipelining

4. Demo

```
In [99]: plt.plot(out_bufferR)

Out[99]: [<matplotlib.lines.Line2D at 0xad960d30>]
```



```
In [100]: plt.plot(out_bufferT)

Out[100]: [<matplotlib.lines.Line2D at 0xacf86af0>]
```