Neuromorphic engineering I

Lab 4: Static Circuits: Current Mirror, Differential Pair, Bump-antibump Circuit

Team member 1: Quillan Favey

Board number: 08

Date: 11.18.22

Lab objectives

The objectives of this lab are to understand and characterize a number of very useful standard static circuits that in subthreshold operation.

The experimental objectives are as follows:

1. To learn how to measure small current using on-chip current-to-frequency (C2F) converter
2. To measure and characterize the differential-pair currents as a function of the input voltages, including the mismatch-caused differential offset voltage.
3. To characterize a bump-antibump circuit and to understand something about its nonidealities.

1 Reading

Read the section on the differential pair, transconductance amplifier, and bump circuit in Chapter 5 of the class book.

2 Prelab

This prelab must be completed before coming to the lab.

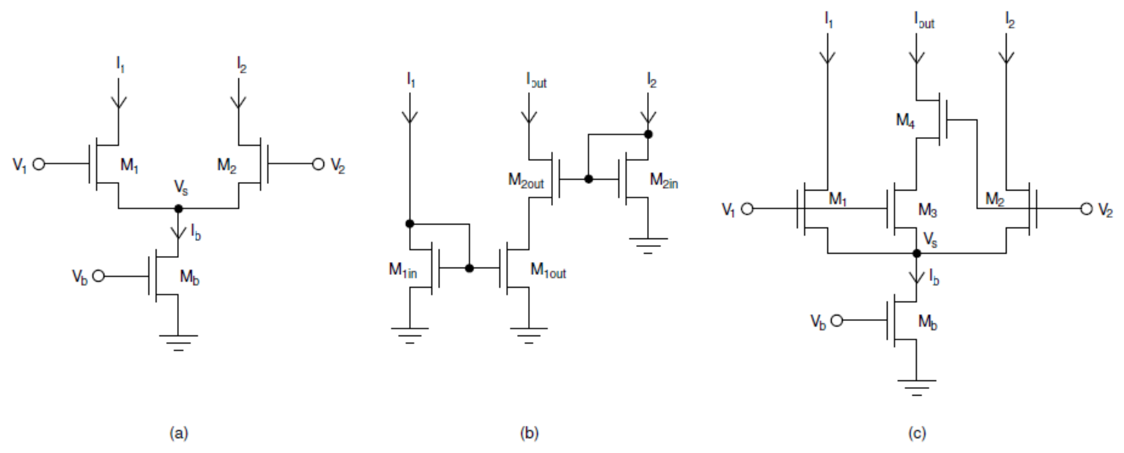


Figure 4.1: (a) Differential pair. (b) Simple current correlator. (c) Bump-antibump circuit.

2.1 Differential pair

All parts of this question refer to the differential pair shown in Fig. 4.1(a). Unless stated otherwise, assume that M_1, M_2, and M_b are in saturation, that they are operated in subthreshold.

- When working with differential circuits, it is often advantageous to express results in terms of the *common mode* voltage (denoted by $\bar{V}$ or V_{cm}) and the *differential mode* voltage (denoted by δV or V_{dm}). These voltages are defined in terms of V_1 and V_2 by $\bar{V} \equiv \frac{1}{2}(V_1 + V_2)$ and $\delta V \equiv V_1 - V_2$. Solve for V_1 and V_2 in terms of $\bar{V}$ and δV.

- $V_2 = \bar{V} - \frac{1}{2}\delta V$
- $V_1 = \bar{V} + \frac{3}{2}\delta V$

- Compute the common source voltage V_s of M_1 and M_2 as a function of the inputs V_1 and V_2, and the bias current I_b.

As seen in the lecture:

- $I_b = I_0 e^{-V_s/U_T}(e^{\kappa V_1/U_T} + e^{\kappa V_2/U_T})$

we can then rearrange to get:

$$\ln I_b = \ln I_0 - \frac{V_s}{U_T} + \ln\left(e^{\frac{\kappa 1}{U_T}} + e^{\frac{v2}{U_T}}\right) \tag{1}$$

$$V_s = U_T\left(\ln\left(\frac{I_0}{I_b}\left(e^{\frac{\kappa V_1}{U_T}} + e^{\frac{\kappa V_2}{U_T}}\right)\right)\right) \tag{2}$$

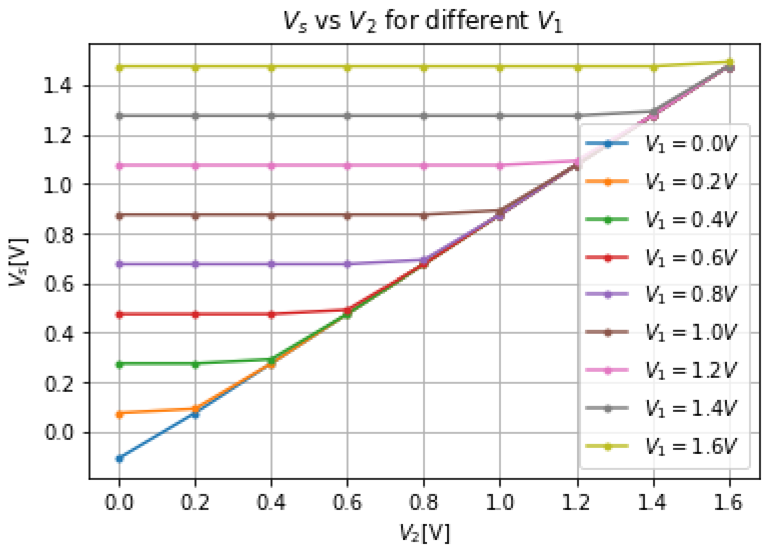
- What restrictions would you put on V_1 and V_2 to ensure that M_b is in saturation?

$$\max(V_1, V_2) > \kappa_n^{-1}(4U_T + \kappa_b V_b)$$

if, $|V_1 - V_2| > 4U_T$

- Holding V_1 constant, sketch V_s versus V_2.

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        UT = 0.025
        I0 = 2e-7
        kappa = 1
        Ib = 3e-5

        V1 = np.arange(0.0, 1.8, 0.2)
        V2 = np.arange(0.0, 1.8, 0.2)
        for V in V1:
            V_s = UT * np.log(I0 / Ib * (np.exp(kappa*V/UT) + np.exp(kappa*V2/UT)))
            plt.plot(V2, V_s, ".-",label=f'$V_1 = {np.round(V,5)}V$')
            plt.xlabel('''$V_2$[V]''')
            plt.ylabel('$V_s$[V]')
        plt.title('$V_s$ vs $V_2$ for different $V_1$')
        plt.legend()
        plt.grid()
        plt.show()
```



- How is the diff-pair related to a source-follower?

The diff pair has the same structure as the source follower, only $I{b} $sharedbyM1andM2whosesourcesareconnectedtothedrainofthebiasMOSFET$(M*{B}$).*

- In what way does V_s approximate the maximum function $\max(V_1, V_2)$? (You will see why this is relevant in the winner-take-all circuit.)

The transistor with a higher Vg will get more current. The only difference is when $V1 \approx V2$ there will be a "sharing of current between the two MOSFETS"

- Compute the currents I_1 and I_2 as a function of V_1, V_2, and I_b.

$$I_1 = I_b \frac{e^{\frac{\kappa v_1}{U_T}}}{e^{\frac{\kappa V_1}{U_T}} + e^{\frac{\kappa V_2}{U_T}}}$$
$$I_2 = I_b \frac{e^{\frac{\kappa V_2}{U_T}}}{e^{\frac{\kappa V_1}{U_T}} + e^{\frac{\kappa V_2}{U_T}}}$$

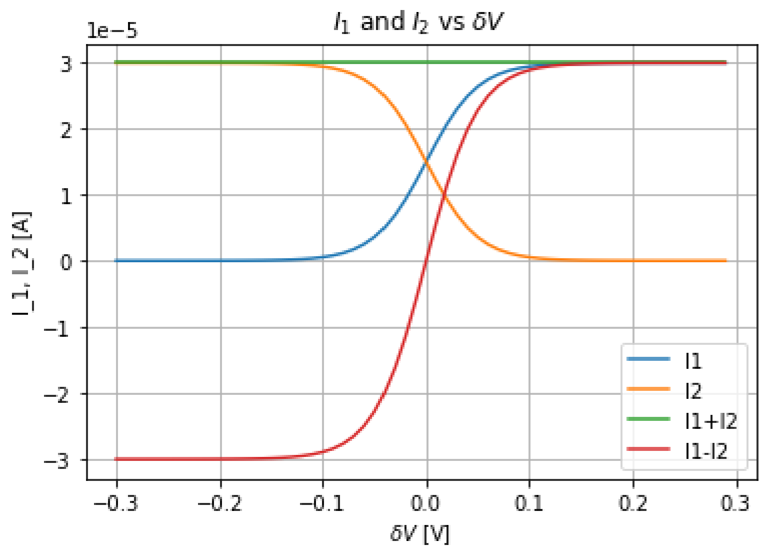
- Now compute the relationship between the differential output current $I_1 - I_2$ and the differential input voltage δV. Remember there is a trick: multiplying by $\exp\left(-\frac{V1+V2}{2}\right)$).

$$I_1 - I_2 = I_b \frac{e^{\frac{\kappa V1}{UT}} - e^{\frac{\kappa V2}{UT}}}{e^{\frac{\kappa V1}{UT}} + e^{\frac{\kappa V2}{UT}}} \; I_1 - I_2 = I_b \tanh(\frac{\kappa}{2UT}(V1 - V2)) \; I_1 - I_2 = I_b \tanh(\frac{\kappa}{UT}\delta V)$$

- Sketch a graph of I_1 and I_2 versus δV. Also sketch the sum $I_1 + I_2$ and the difference $I_1 - I_2$ on the same axes.

In [ ]:
```
V1 = np.arange(0.0,0.6,0.01)
V2 = 0.3
kappa = 1
UT = 0.025
delta_V = V1-V2
I1 = Ib*np.exp(kappa*V1/UT)/(np.exp(kappa*V1/UT) + np.exp(kappa*V2/UT))
I2 = Ib*np.exp(kappa*V2/UT)/(np.exp(kappa*V1/UT) + np.exp(kappa*V2/UT))

plt.plot(delta_V,I1,label="I1")
plt.plot(delta_V,I2,label="I2")
plt.plot(delta_V,I1+I2,label="I1+I2")
plt.plot(delta_V,I1-I2,label="I1-I2")
plt.grid()
plt.xlabel('$\delta V$ [V]')
plt.ylabel('I_1, I_2 [A]')
plt.title('$I_1$ and $I_2$ vs $\delta V$')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x26fcafd03d0>
```



2.2 Current correlator

For the simple current correlator in Fig. 4.1(b).

- Show that $I_{out} = \frac{r_1 I_1 r_2 I_2}{r_1 I_1 + r_2 I_2}$, where r_1 and r_2 denote the W/L ratios for the transistors connected to V_1 and V_2 respectively. This means that $r_1 = \frac{w_{1out}}{w_{1in}}$ and $r_2 = \frac{w_{2out}}{w_{2in}}$, where the w's denote the W/L ratios of the corresponding transistors. Assume that M_{2out} is in saturation, but note that M_{1out} may not be.

$$I = Se^{-V_s}\frac{e^{V_1}e^{V_2}}{e^{V_1}+e^{V_2}}$$
$$= S\frac{I_1 I_2}{I_1+I_2}.$$

- Let $I_1 = \frac{I_t}{2}(1+x)$, $I_2 = \frac{I_t}{2}(1-x)$, where $I_t \equiv I_1 + I_2$ is the total input current and $x \equiv \frac{I_1 - I_2}{I_t}$ is a dimensionless difference current.

(a) Substitute these expressions into the espression for I_{out} in exercise 2 and obtain an expression for I_{out} in terms of I_t and x.

$$I_{out} = \frac{r_1 I_1 r_2 I_2}{r_1 I_1 + r_2 I_2} \tag{3}$$

$$I_{\text{out}} = \frac{r_1 \frac{I_t}{2}(1+x) r_2 \frac{I_t}{2}(1-x)}{r_1 \frac{I_t}{2}(1+x) + r_2 \frac{I_t}{2}(1-x)} \tag{4}$$

After simplifying a bit we get:

$$I_{\text{out}} = \frac{r_1 r_2 I_t \left(1 - x^2\right)}{2\left(r_1(1+x) + r_2(1-x)\right)} \tag{5}$$

(b) Simplify your result assuming $r_1 = r_2 \equiv r$ and sketch a graph of I_{out} vs. x. How is the graph modified if $r_1 > r_2$?

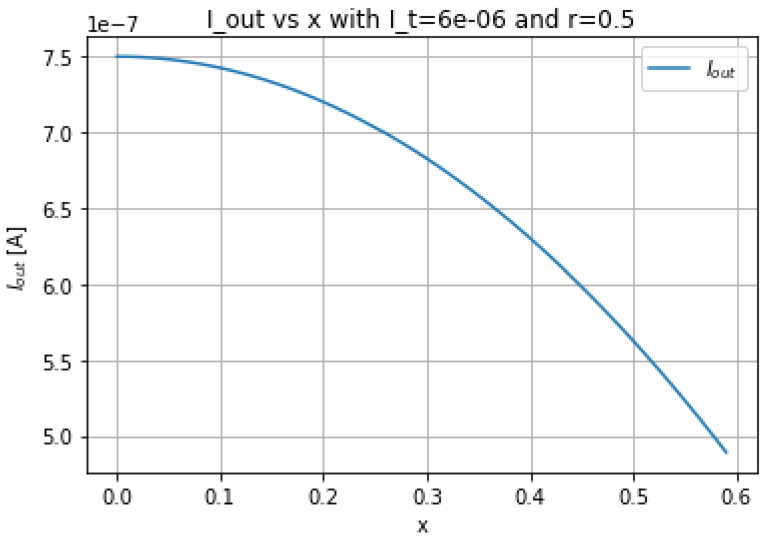
$$I_{\text{out}} = \frac{r^2 I_t \left(1 - x^2\right)}{2r\left((1+x) + (1-x)\right)} \tag{6}$$

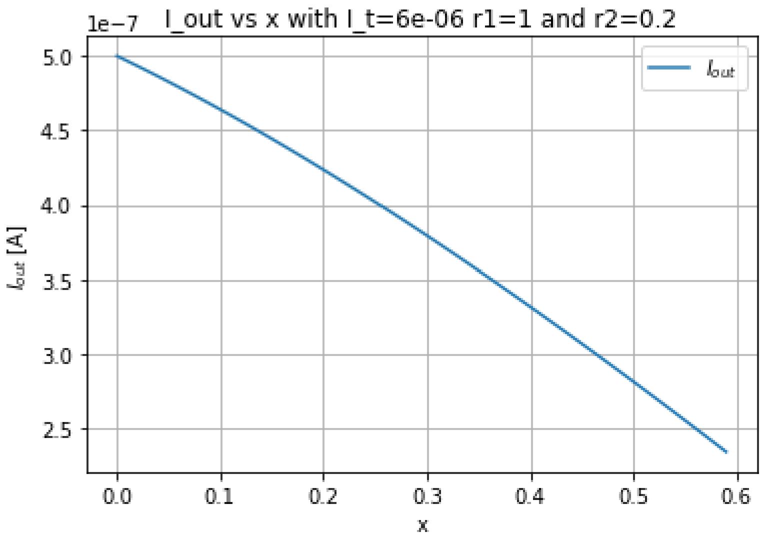
$$I_{\text{out}} = \frac{r^2 I_t \left(1 - x^2\right)}{4r} \tag{7}$$

$$I_{\text{out}} = \frac{r I_t \left(1 - x^2\right)}{4} \tag{8}$$

In [ ]:

```
x = np.arange(0.0, 0.6, 0.01)
I_t = 0.6e-5
r = 0.5
I_out = 1/4*r*I_t*(1-x**2)
plt.plot(x, I_out, label='$I_{out}$')
plt.title(f'I_out vs x with I_t={I_t} and r={r}')
plt.xlabel('x')
plt.ylabel('$I_{out}$ [A]')
plt.legend()
plt.grid()
plt.show()
#r1>r2
r1 = 1
r2 = 0.2
I_out = r1*r2*I_t*(1-x**2)/(2*(r1*(1+x)+r2*(1-x)))
plt.plot(x, I_out, label='$I_{out}$')
plt.title(f'I_out vs x with I_t={I_t} r1={r1} and r2={r2}')
plt.xlabel('x')
plt.ylabel('$I_{out}$ [A]')
plt.legend()
plt.grid()
plt.show()
```





(c) Show that if I_1 and I_2 are generated by a differential pair (see earlier question) then $x = \tanh\left(\frac{\kappa(V_1 - V_2)}{2U_T}\right)$ and I_t is the differential pair's bias current.

$$x = \frac{I_1 - I_2}{I_t} \tag{9}$$

$$x = \frac{I_b \tanh(\frac{\kappa}{UT}\delta V)}{I_t} \tag{10}$$

as $I_b \equiv I_t$

$$x = \tanh(\frac{\kappa}{UT}\delta V) \tag{11}$$

$$x = \tanh(\frac{\kappa}{2UT}(V1 - V2)) \tag{12}$$

2.2 Bump-antibump circuit

Now consider the bump-antibump circuit shown in Fig. 4.1(c).

- Assume that $r1 = r2 \equiv r$ and $x = \tanh\left(\frac{\kappa(V_1-V_2)}{2U_T}\right)$. Compute I_{out} in terms of x, r, and I_b by substituting $I_t = I_b - I_{out}$ in the equation for I_{out} in exercise 2 and solving for I_{out} .

$$\begin{aligned}
I_{\text{out}} &= \frac{1}{4} r I_t \left(1-x^2\right) \\
I_{\text{out}} &= \frac{1}{4} r \left(I_b - I_{\text{out}}\right)\left(1-x^2\right) \\
I_{\text{out}} &= \frac{1/4 r I_b (1-x^2)}{(1+1/4+r(1-x^2))} \\
I_{\text{out}} &= \frac{r I_b (1-x^2)}{(4+r(1-x^2))}
\end{aligned}$$

- Express your result in terms of the hyperbolic cosine function (cosh). You may want to use the hyperbolic function relationships

```
\begin{equation}
\cosh^2(x)-\sinh^2(x)=1
\end{equation}
\\
\\
\begin{equation}
\tanh^2(x)=1-\frac{1}{\cosh^2(x)}
\end{equation}
```

You should end up with the result $I_{out} = \frac{I_b}{1+\frac{4}{r}\cosh^2(\frac{\kappa\Delta V}{2U_T})}$

$$\begin{aligned}
I_{\text{out}} &= \frac{r I_b (1-x^2)}{(4+r(1-x^2))} \\
I_{\text{out}} &= \frac{r I_b (1-(\tanh\left(\frac{\kappa(V_1-V_2)}{2U_T}\right)^2)}{(4+r(1-(\tanh\left(\frac{\kappa(V_1-V_2)}{2U_T}\right)^2))}
\end{aligned}$$

$$I_{out} = \frac{I_b}{1+\frac{4}{r}\cosh^2(\frac{\kappa\Delta V}{2U_T})} \tag{13}$$

- What fraction of I_b will flow down the middle branch (the bump branch) if $V_1 = V_2$?

If $V_1 = V_2$ we get $\Delta V = 0$, and so $cosh^2(0) = 1$. We get the following:

$$\begin{aligned}
I_{out} &= \frac{I_b}{1+\frac{4}{r}} \\
I_{out} &= I_b \frac{r}{r+4}
\end{aligned}$$

- Does the bump-antibump circuit compute "soft" or analog logic operations AND and XOR between the two voltage inputs V_1 and V_2?

If we look at I1 and I2 as our outputs, we can clearly see (refering to the output characteristics graph from the book) a XOR like operation but with the intermediate states when approaching to V1 = V2.The same can be said when looking at the Imid (or Iout) and the AND operation

4 Setup

4.1 Connect the device

```
In [ ]: # import the necessary library to communicate with the hardware
        import pyplane
```

```
In [ ]: # create a Plane object and open the communication
        if 'p' not in locals():
            p = pyplane.Plane()
            try:
                p.open('/dev/ttyACM0') # Open the USB device ttyACM0 (the board).
            except RuntimeError as e:
                print(e)

        # Note that if you plug out and plug in the USB device in a short time interval, the c
        # then you may get error messages with open(...ttyACM0). So please avoid frenquently p
```

```
In [ ]: p.get_firmware_version()

Out[ ]: (1, 8, 8)
```

```
In [ ]: # Send a reset signal to the board, check if the LED blinks
        p.reset(pyplane.ResetType.Soft)

Out[ ]: <TeensyStatus.Success: 0>
```

```
In [ ]: # NOTE: You must send this request events every time you do a reset operetion, otherwi
        # Because the class chip need to handshake with some other devices to get the communic
        p.request_events(1)
```

```
In [ ]: # Try to read something, make sure the chip responses
        p.read_current(pyplane.AdcChannel.GO0_N)

Out[ ]: 4.833984235119715e-08
```

4.2 Select the multiplexer and demultiplexer

You may remember that in the last two labs, before we measure N-FET or P-FET we had to send a configuration event first. That is because pin number has always been a bottleneck for IC design and we could not make a gigantic chip with hundreds of pins. But on the other hand, the transistors are so tiny that we could put yet a lot more, so we decided to make some of the circuits share some input-output pins and C2F channels using analog mux/demux. For more details please refer to the chip documentation (not needed for the lab).

4.3 Bias Generator (BiasGen or BG)

For any analog circuit, you may need to set some fixed currents/voltages in order to put all transistors in the desired operation regime, which are called biases (e.g. I_b). Since there are hundreds of biases on our chip that need to be set at the same time, it is impossible to just use a demultiplexer (as what we were doing when measuring N-FET and P-FET in the previous labs). The way we are doing it (and also the way most neuromorphic chips work) is by having a so-called *Bias Generator* (or *BiasGen* in short) circuit, that outputs a current that can be divided and mirrored to each individual circuit. In a simplified form, the output of a branch of the BiasGen will be the gate voltage V_b for the bias current I_b, and if the current mirror has a ratio of w and the bias transistor operates in subthreshold-saturation:

$$I_b = w \frac{BG_{fine}}{256} I_{BG_{master}} \tag{14}$$

Where $I_{BG_{master}}$ is the `BiasGenMasterCurrent` $\in \{60 \text{ pA}, 460 \text{ pA}, 3.8 \text{ nA}, 30 \text{ nA}, 240 \text{ nA}\}$, BG_{fine} is the integer fine value $\in [0, 256)$

To set a bias, use the funcion similar to the following (see 4.4 for examples):

```
p.send_coach_event(pyplane.Coach.generate_biasgen_event(\

pyplane.Coach.BiasAddress.BIAS_NAME_STARTS_WITH_THREE_LETTER_CIRCUIT_NAME,
 \
    pyplane.Coach.BiasType.MATCH_LAST_CHAR_OF_BIAS_NAME, \
    pyplane.Coach.BiasGenMasterCurrent.MASTER_CURRENT, FINE_VALUE))
```

4.4 C2F circuit

To measure very small current (in our case from 1 pA to 10 nA), a very widely used method is called current-to-frequency conversion. The output frequency f can be expressed as a function of input current I:

$$f = \frac{I}{C\Delta U} \tag{15}$$

where C is a capactance which is charged by the input current and ΔU is difference of the reference voltages where the circuit resets. For more details please refer to the chip documentation (not needed for the lab).

- To set up the C2F circuit, you have to set the following biases:

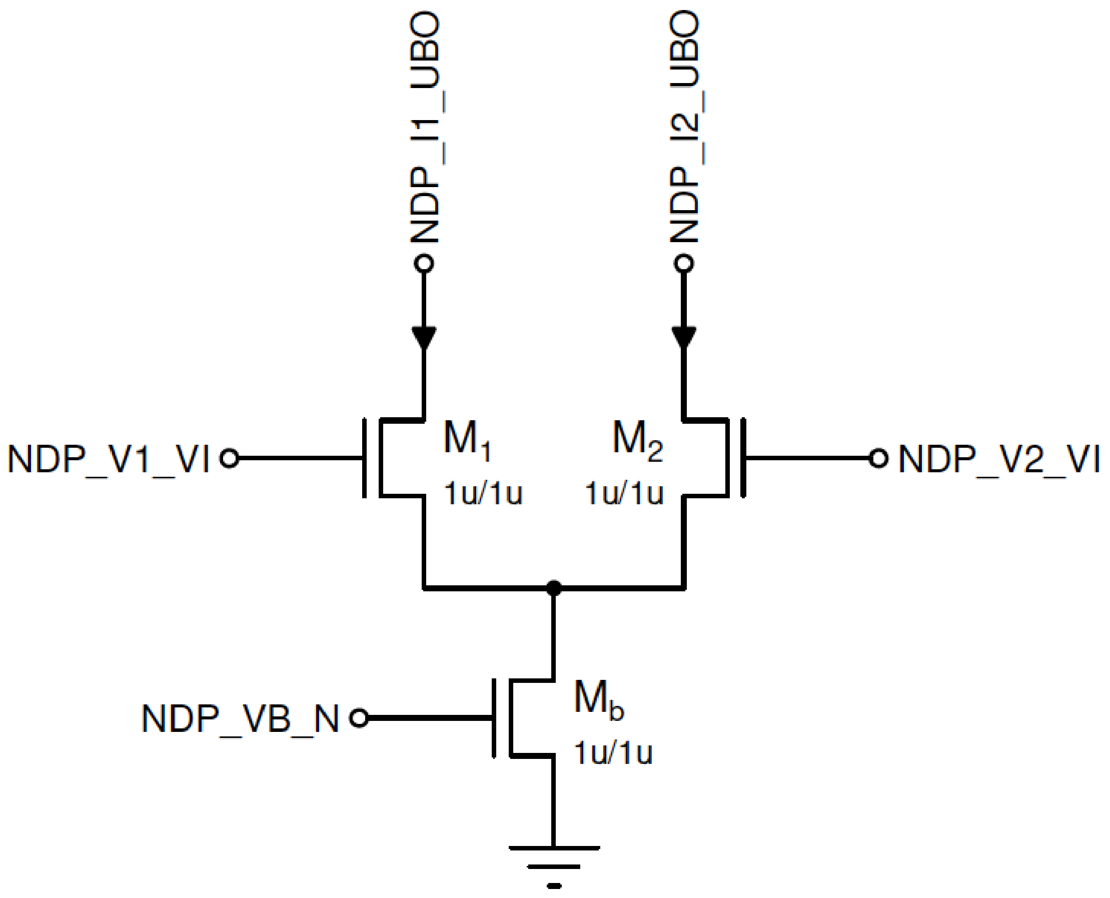
In [ ]:
```
p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.C2F_HYS_P, \
    pyplane.Coach.BiasType.P, \
    pyplane.Coach.BiasGenMasterCurrent.I60pA, 100)])

p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.C2F_BIAS_P, \
    pyplane.Coach.BiasType.P, \
    pyplane.Coach.BiasGenMasterCurrent.I240nA, 255)])

p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.C2F_PWLK_P, \
    pyplane.Coach.BiasType.P, \
    pyplane.Coach.BiasGenMasterCurrent.I240nA, 255)])

p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.C2F_REF_L, \
    pyplane.Coach.BiasType.N, \
    pyplane.Coach.BiasGenMasterCurrent.I30nA, 255)])

p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.C2F_REF_H, \
    pyplane.Coach.BiasType.P, \
    pyplane.Coach.BiasGenMasterCurrent.I30nA, 255)])
```

- The output of the C2F circuit is a bunch of *events* that will be counted by the Teensy microcontroller and sent to the PC.

5 N-FET differential pair circuit (NDP)

In this experiment you will measure the dependence of the differential pair currents I_1 and I_2 on the differential input voltage V_{diff}.

5.0 Schematic and pin map



I_1 = **NDP_I1_UBO = C2F[0]**

I_2 = **NDP_I2_UBO = C2F[1]**

V_1 = **NDP_V1_VI = AIN5**

V_2 = **NDP_V2_VI = AIN6**

5.1 Chip configuration

In [ ]:
```
p.send_coach_events([pyplane.Coach.generate_aerc_event( \
    pyplane.Coach.CurrentOutputSelect.SelectLine5, \
    pyplane.Coach.VoltageOutputSelect.NoneSelected, \
    pyplane.Coach.VoltageInputSelect.SelectLine2, \
    pyplane.Coach.SynapseSelect.NoneSelected, 0)])
```

5.2 C2F calibration

Assume the W/L ratio between the differential pair bias transistor Mb and the BiasGen output transistor is **1**.

- If we trust the value for I_b calculated from the formula in 4.3, how do we find out the mapping between I and f for each C2F channel? (Hint: what is I_1 (I_2) when $V_1 \gg (\ll) V_2$?)

Using KCL it can be inferred that

$$I_1 + I_2 = I_b \Rightarrow I_1 = I_b - I_2.$$

From the prelab it is also known that

$$I_2 = I_b \frac{e^{\frac{\kappa}{U_T}V_2}}{e^{\frac{\kappa}{U_T}V_1} + e^{\frac{\kappa}{U_T}V_2}}.$$

Therefore

$$I_1 = I_b \left(1 - \frac{e^{\frac{\kappa}{U_T}V_2}}{e^{\frac{\kappa}{U_T}V_1} + e^{\frac{\kappa}{U_T}V_2}}\right).$$

When $V_1 \gg V_2$

$$I_1(I_2) = I_b \left(1 - \frac{e^{\frac{\kappa}{U_T}V_2}}{\underbrace{e^{\frac{\kappa}{U_T}V_1} + e^{\frac{\kappa}{U_T}V_2}}_{\approx 0}}\right) \approx I_b.$$

Using the equation in 4.3 thus yields f as

$$\Rightarrow f_1 = \frac{I_1}{C\Delta U} = \frac{I_b}{C\Delta U}.$$

When $V_1 \ll V_2$

$$I_1(I_2) = I_b \left(1 - \frac{e^{\frac{\kappa}{U_T}V_2}}{\underbrace{e^{\frac{\kappa}{U_T}V_1} + e^{\frac{\kappa}{U_T}V_2}}_{\approx 1}}\right) \approx 0.$$

Therefore,

$$\Rightarrow f_1 = \frac{I_1}{C\Delta U} \approx 0$$

in this case.

From these results, it can be concluded that the mapping between I_1 and f_1 can be obtained by evaluating $V_1 \gg V_2$ and measuring f_1 over a range I_b.

5.2.1 Calibrate C2F response for I1

- Set fixed voltages for V_1 and V_2

In [ ]:
```
p.set_voltage(pyplane.DacChannel.AIN5,0.6) # V1 = 0.6
p.set_voltage(pyplane.DacChannel.AIN6,0.2) # V2 = 0.2
```

Out[ ]:
```
0.19882699847221375
```

Choose values such that $V_1 \gg V_2$.

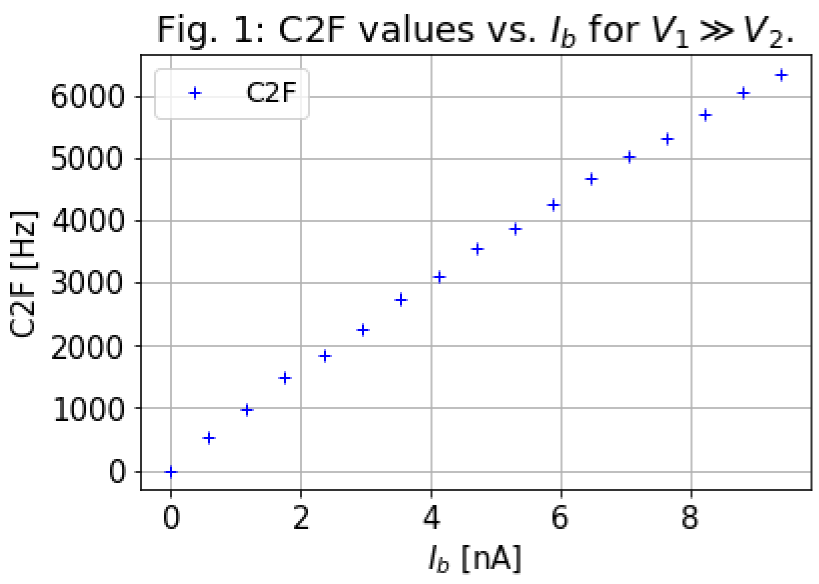
- Data aquisition (Hint: linear range $I \leq 10$ nA)
- You can follow the example below

In [ ]:
```
import pyplane
import numpy as np
import time
import matplotlib.pyplot as plt
# your code

bg_fine_calI1 = np.arange(0,85,5) # bias current sweep range

c2f_calI1 = []

for n in range(len(bg_fine_calI1)):

    # set bias
    p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.NDP_VB_N, \
    pyplane.Coach.BiasType.N, \
    pyplane.Coach.BiasGenMasterCurrent.I30nA, bg_fine_calI1[n])])

    time.sleep(0.5) # settle time

    # read c2f values for 0.1s duration
    c2f_calI1_temp = p.read_c2f_output(0.1)
    c2f_calI1.append(c2f_calI1_temp[0])
print(c2f_calI1)

np.savetxt('c2f_calI1_vs_bg_fine_calI1.csv',[c2f_calI1,bg_fine_calI1], delimiter=',')
```

```
[2, 528, 996, 1481, 1864, 2281, 2732, 3113, 3544, 3882, 4250, 4669, 5034, 5310, 5686,
6065, 6349]
```

- Plot C2F value vs Ib
- You can follow the example below (but remember to save data firstly)

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
plt.rcParams.update({'font.size': 15})

c2f_calI2,bg_fine_calI1 = np.loadtxt('c2f_calI1_vs_bg_fine_calI1.csv', delimiter=',')

Ib_calI2 = bg_fine_calI1/256*30

plt.plot(Ib_calI2,c2f_calI2,'b+')

plt.xlabel('$I_b$ [nA]')
plt.ylabel('C2F [Hz]')
plt.legend(['C2F'],prop={'size': 14})
plt.title('Fig. 1: C2F values vs. $I_b$ for $V_1 \gg V_2$.')
plt.grid()
plt.show()
```



- Save data
- You can follow the example below

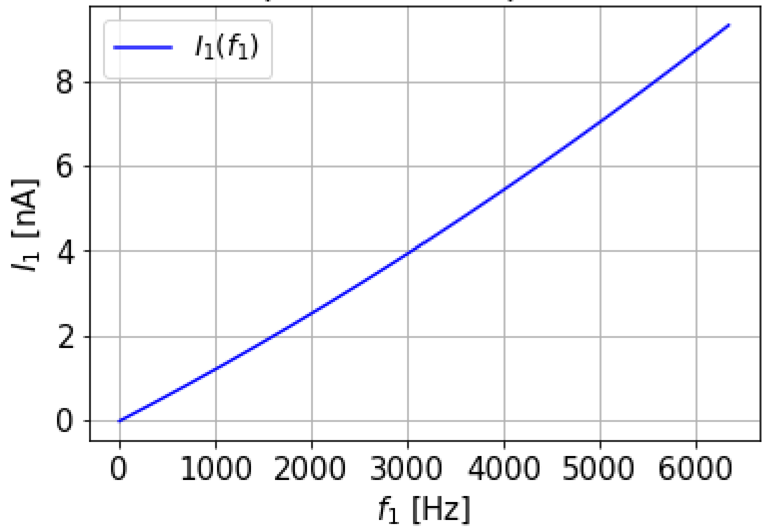
In [ ]:
```
# if the data looks nice, save it!
data_I1cal = [c2f_calI2,bg_fine_calI1]
# save to csv file
np.savetxt('c2f_calI1_vs_bg_fine_calI1.csv', data_I1cal, delimiter=',')
```

- Extract the function $I_1(f_1)$ (Hint: use higher order polynomial to increase accuracy)
- You can follow the example below

In [ ]:
```
# fit quadratic polynomial to C2F vs Ib data
a2_I1cal,a1_I1cal,a0_I1cal = np.polyfit(c2f_calI2[:16],Ib_calI2[:16],2)

print(a0_I1cal)
print(a1_I1cal)
print(a2_I1cal)

range_I1cal = np.arange(1,c2f_calI2[16],14) # select interpolation interval, omitting
print(c2f_calI2[16])
print(range_I1cal)
plt.plot(range_I1cal,a0_I1cal+a1_I1cal*range_I1cal+a2_I1cal*range_I1cal**2,'b-')

plt.xlabel('$f_1$ [Hz]')
plt.ylabel('$I_1$ [nA]')
plt.legend(['$I_1(f_1)$'],prop={'size': 14})
plt.title('Fig. 2: Quadratic interpolation of $I_1$ plotted as a function of $f_1$. ')
plt.grid()
plt.show()
```

```
-0.034814858933916705
0.0011778400324940238
4.686916700881777e-08
6349.0
[1.000e+00 1.500e+01 2.900e+01 4.300e+01 5.700e+01 7.100e+01 8.500e+01
 9.900e+01 1.130e+02 1.270e+02 1.410e+02 1.550e+02 1.690e+02 1.830e+02
 1.970e+02 2.110e+02 2.250e+02 2.390e+02 2.530e+02 2.670e+02 2.810e+02
 2.950e+02 3.090e+02 3.230e+02 3.370e+02 3.510e+02 3.650e+02 3.790e+02
 3.930e+02 4.070e+02 4.210e+02 4.350e+02 4.490e+02 4.630e+02 4.770e+02
 4.910e+02 5.050e+02 5.190e+02 5.330e+02 5.470e+02 5.610e+02 5.750e+02
 5.890e+02 6.030e+02 6.170e+02 6.310e+02 6.450e+02 6.590e+02 6.730e+02
 6.870e+02 7.010e+02 7.150e+02 7.290e+02 7.430e+02 7.570e+02 7.710e+02
 7.850e+02 7.990e+02 8.130e+02 8.270e+02 8.410e+02 8.550e+02 8.690e+02
 8.830e+02 8.970e+02 9.110e+02 9.250e+02 9.390e+02 9.530e+02 9.670e+02
 9.810e+02 9.950e+02 1.009e+03 1.023e+03 1.037e+03 1.051e+03 1.065e+03
 1.079e+03 1.093e+03 1.107e+03 1.121e+03 1.135e+03 1.149e+03 1.163e+03
 1.177e+03 1.191e+03 1.205e+03 1.219e+03 1.233e+03 1.247e+03 1.261e+03
 1.275e+03 1.289e+03 1.303e+03 1.317e+03 1.331e+03 1.345e+03 1.359e+03
 1.373e+03 1.387e+03 1.401e+03 1.415e+03 1.429e+03 1.443e+03 1.457e+03
 1.471e+03 1.485e+03 1.499e+03 1.513e+03 1.527e+03 1.541e+03 1.555e+03
 1.569e+03 1.583e+03 1.597e+03 1.611e+03 1.625e+03 1.639e+03 1.653e+03
 1.667e+03 1.681e+03 1.695e+03 1.709e+03 1.723e+03 1.737e+03 1.751e+03
 1.765e+03 1.779e+03 1.793e+03 1.807e+03 1.821e+03 1.835e+03 1.849e+03
 1.863e+03 1.877e+03 1.891e+03 1.905e+03 1.919e+03 1.933e+03 1.947e+03
 1.961e+03 1.975e+03 1.989e+03 2.003e+03 2.017e+03 2.031e+03 2.045e+03
 2.059e+03 2.073e+03 2.087e+03 2.101e+03 2.115e+03 2.129e+03 2.143e+03
 2.157e+03 2.171e+03 2.185e+03 2.199e+03 2.213e+03 2.227e+03 2.241e+03
 2.255e+03 2.269e+03 2.283e+03 2.297e+03 2.311e+03 2.325e+03 2.339e+03
 2.353e+03 2.367e+03 2.381e+03 2.395e+03 2.409e+03 2.423e+03 2.437e+03
 2.451e+03 2.465e+03 2.479e+03 2.493e+03 2.507e+03 2.521e+03 2.535e+03
 2.549e+03 2.563e+03 2.577e+03 2.591e+03 2.605e+03 2.619e+03 2.633e+03
 2.647e+03 2.661e+03 2.675e+03 2.689e+03 2.703e+03 2.717e+03 2.731e+03
 2.745e+03 2.759e+03 2.773e+03 2.787e+03 2.801e+03 2.815e+03 2.829e+03
 2.843e+03 2.857e+03 2.871e+03 2.885e+03 2.899e+03 2.913e+03 2.927e+03
 2.941e+03 2.955e+03 2.969e+03 2.983e+03 2.997e+03 3.011e+03 3.025e+03
 3.039e+03 3.053e+03 3.067e+03 3.081e+03 3.095e+03 3.109e+03 3.123e+03
 3.137e+03 3.151e+03 3.165e+03 3.179e+03 3.193e+03 3.207e+03 3.221e+03
 3.235e+03 3.249e+03 3.263e+03 3.277e+03 3.291e+03 3.305e+03 3.319e+03
 3.333e+03 3.347e+03 3.361e+03 3.375e+03 3.389e+03 3.403e+03 3.417e+03
 3.431e+03 3.445e+03 3.459e+03 3.473e+03 3.487e+03 3.501e+03 3.515e+03
 3.529e+03 3.543e+03 3.557e+03 3.571e+03 3.585e+03 3.599e+03 3.613e+03
 3.627e+03 3.641e+03 3.655e+03 3.669e+03 3.683e+03 3.697e+03 3.711e+03
 3.725e+03 3.739e+03 3.753e+03 3.767e+03 3.781e+03 3.795e+03 3.809e+03
 3.823e+03 3.837e+03 3.851e+03 3.865e+03 3.879e+03 3.893e+03 3.907e+03
 3.921e+03 3.935e+03 3.949e+03 3.963e+03 3.977e+03 3.991e+03 4.005e+03
 4.019e+03 4.033e+03 4.047e+03 4.061e+03 4.075e+03 4.089e+03 4.103e+03
 4.117e+03 4.131e+03 4.145e+03 4.159e+03 4.173e+03 4.187e+03 4.201e+03
 4.215e+03 4.229e+03 4.243e+03 4.257e+03 4.271e+03 4.285e+03 4.299e+03
 4.313e+03 4.327e+03 4.341e+03 4.355e+03 4.369e+03 4.383e+03 4.397e+03
 4.411e+03 4.425e+03 4.439e+03 4.453e+03 4.467e+03 4.481e+03 4.495e+03
 4.509e+03 4.523e+03 4.537e+03 4.551e+03 4.565e+03 4.579e+03 4.593e+03
 4.607e+03 4.621e+03 4.635e+03 4.649e+03 4.663e+03 4.677e+03 4.691e+03
 4.705e+03 4.719e+03 4.733e+03 4.747e+03 4.761e+03 4.775e+03 4.789e+03
 4.803e+03 4.817e+03 4.831e+03 4.845e+03 4.859e+03 4.873e+03 4.887e+03
 4.901e+03 4.915e+03 4.929e+03 4.943e+03 4.957e+03 4.971e+03 4.985e+03
 4.999e+03 5.013e+03 5.027e+03 5.041e+03 5.055e+03 5.069e+03 5.083e+03
 5.097e+03 5.111e+03 5.125e+03 5.139e+03 5.153e+03 5.167e+03 5.181e+03
 5.195e+03 5.209e+03 5.223e+03 5.237e+03 5.251e+03 5.265e+03 5.279e+03
 5.293e+03 5.307e+03 5.321e+03 5.335e+03 5.349e+03 5.363e+03 5.377e+03
 5.391e+03 5.405e+03 5.419e+03 5.433e+03 5.447e+03 5.461e+03 5.475e+03
 5.489e+03 5.503e+03 5.517e+03 5.531e+03 5.545e+03 5.559e+03 5.573e+03
 5.587e+03 5.601e+03 5.615e+03 5.629e+03 5.643e+03 5.657e+03 5.671e+03
 5.685e+03 5.699e+03 5.713e+03 5.727e+03 5.741e+03 5.755e+03 5.769e+03
 5.783e+03 5.797e+03 5.811e+03 5.825e+03 5.839e+03 5.853e+03 5.867e+03
 5.881e+03 5.895e+03 5.909e+03 5.923e+03 5.937e+03 5.951e+03 5.965e+03
 5.979e+03 5.993e+03 6.007e+03 6.021e+03 6.035e+03 6.049e+03 6.063e+03
 6.077e+03 6.091e+03 6.105e+03 6.119e+03 6.133e+03 6.147e+03 6.161e+03
 6.175e+03 6.189e+03 6.203e+03 6.217e+03 6.231e+03 6.245e+03 6.259e+03
 6.273e+03 6.287e+03 6.301e+03 6.315e+03 6.329e+03 6.343e+03]
```

Fig. 2: Quadratic interpolation of I_1 plotted as a function of f_1.



5.2.2 Calibration C2F response for I2

- Set vixed voltages for V_1 and V_2

In [ ]:
```
p.set_voltage(pyplane.DacChannel.AIN5,0.2) # V1 = 0.2
p.set_voltage(pyplane.DacChannel.AIN6,0.6) # V2 = 0.6
```

Out[ ]:
```
0.5982405543327332
```

Choose values such that $V_1 \ll V_2$.

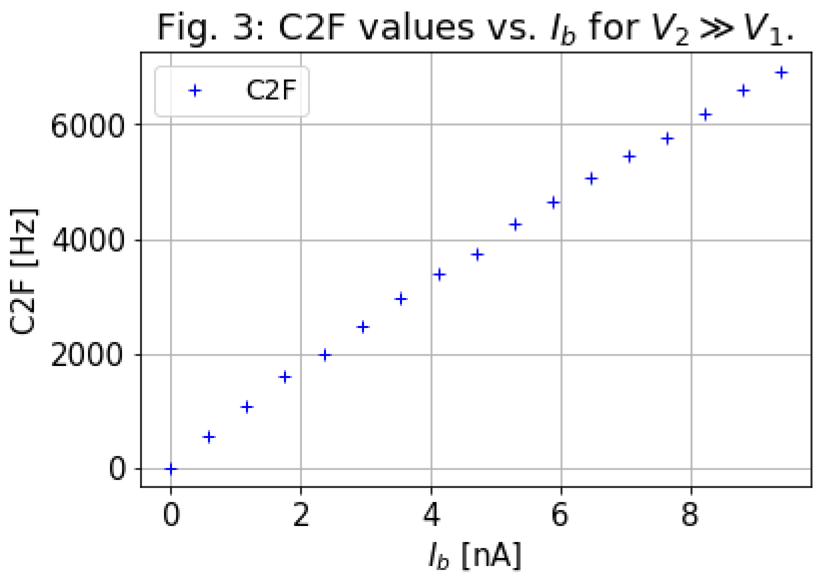
- Data aquisition (Hint: linear range $I \leq 10$ nA)

In [ ]:
```
import pyplane
import numpy as np
import time
import matplotlib.pyplot as plt
# your code

bg_fine_calI2 = np.arange(0,85,5) # bias current sweep range

c2f_calI2 = []

for n in range(len(bg_fine_calI2)):

    # set bias
    p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.NDP_VB_N, \
    pyplane.Coach.BiasType.N, \
    pyplane.Coach.BiasGenMasterCurrent.I30nA, bg_fine_calI2[n])])

    time.sleep(0.5) # settle time

    # read c2f values for 0.1s duration
    c2f_calI2_temp = p.read_c2f_output(0.1)
    c2f_calI2.append(c2f_calI2_temp[1])   #set index for c2F to 1
print(c2f_calI2)

np.savetxt('c2f_calI2_vs_bg_fine_calI2.csv',[c2f_calI2,bg_fine_calI2], delimiter=',')
```

```
[3, 561, 1079, 1601, 1990, 2487, 2976, 3398, 3741, 4257, 4634, 5080, 5469, 5777, 620
0, 6601, 6934]
```

- Plot

In [ ]:
```
plt.rcParams.update({'font.size': 15})

c2f_calI2,bg_fine_calI2 = np.loadtxt('c2f_calI2_vs_bg_fine_calI2.csv', delimiter=',')

Ib_calI2 = bg_fine_calI2/256*30

plt.plot(Ib_calI2,c2f_calI2,'b+')

plt.xlabel('$I_b$ [nA]')
plt.ylabel('C2F [Hz]')
plt.legend(['C2F'],prop={'size': 14})
plt.title('Fig. 3: C2F values vs. $I_b$ for $V_2 \gg V_1$.')
plt.grid()
plt.show()
```



- Save data

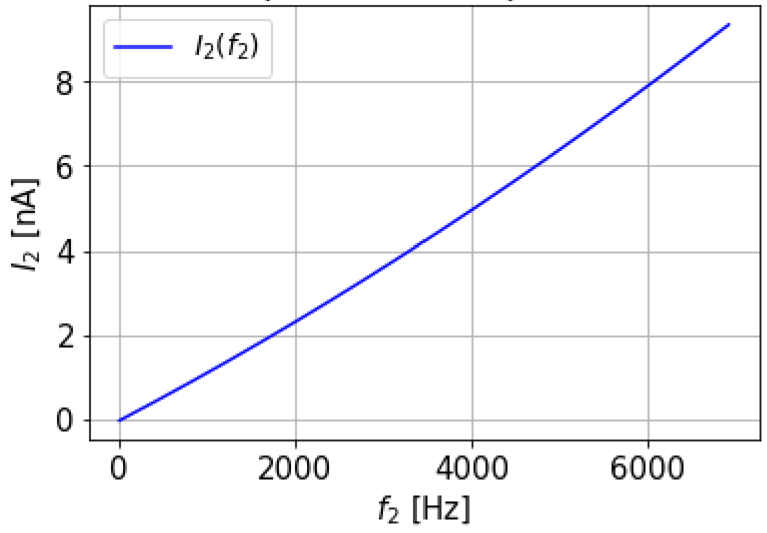
In [ ]:
```
#already saved previously
```

- Extract the function $I_2\left(f_2\right)$ (Hint: use higher order polynomial to increase accuracy)

In [ ]:
```
# fit quadratic polynomial to C2F vs Ib data
a2_I2cal,a1_I2cal,a0_I2cal = np.polyfit(c2f_calI2[:16],Ib_calI2[:16],2)

print(a0_I2cal)
print(a1_I2cal)
print(a2_I2cal)

range_I2cal = np.arange(1,c2f_calI2[16],14) # select interpolation interval, omitting
print(c2f_calI2[16])
print(range_I2cal)
plt.plot(range_I2cal,a0_I2cal+a1_I2cal*range_I2cal+a2_I2cal*range_I2cal**2,'b-')

plt.xlabel('$f_2$ [Hz]')
plt.ylabel('$I_2$ [nA]')
plt.legend(['$I_2(f_2)$'],prop={'size': 14})
plt.title('Fig. 4: Quadratic interpolation of $I_2$ plotted as a function of $f_2$. ')
plt.grid()
plt.show()
```

```
-0.032601584237505546
0.0010950847918774383
3.729204124908894e-08
6934.0
[1.000e+00 1.500e+01 2.900e+01 4.300e+01 5.700e+01 7.100e+01 8.500e+01
 9.900e+01 1.130e+02 1.270e+02 1.410e+02 1.550e+02 1.690e+02 1.830e+02
 1.970e+02 2.110e+02 2.250e+02 2.390e+02 2.530e+02 2.670e+02 2.810e+02
 2.950e+02 3.090e+02 3.230e+02 3.370e+02 3.510e+02 3.650e+02 3.790e+02
 3.930e+02 4.070e+02 4.210e+02 4.350e+02 4.490e+02 4.630e+02 4.770e+02
 4.910e+02 5.050e+02 5.190e+02 5.330e+02 5.470e+02 5.610e+02 5.750e+02
 5.890e+02 6.030e+02 6.170e+02 6.310e+02 6.450e+02 6.590e+02 6.730e+02
 6.870e+02 7.010e+02 7.150e+02 7.290e+02 7.430e+02 7.570e+02 7.710e+02
 7.850e+02 7.990e+02 8.130e+02 8.270e+02 8.410e+02 8.550e+02 8.690e+02
 8.830e+02 8.970e+02 9.110e+02 9.250e+02 9.390e+02 9.530e+02 9.670e+02
 9.810e+02 9.950e+02 1.009e+03 1.023e+03 1.037e+03 1.051e+03 1.065e+03
 1.079e+03 1.093e+03 1.107e+03 1.121e+03 1.135e+03 1.149e+03 1.163e+03
 1.177e+03 1.191e+03 1.205e+03 1.219e+03 1.233e+03 1.247e+03 1.261e+03
 1.275e+03 1.289e+03 1.303e+03 1.317e+03 1.331e+03 1.345e+03 1.359e+03
 1.373e+03 1.387e+03 1.401e+03 1.415e+03 1.429e+03 1.443e+03 1.457e+03
 1.471e+03 1.485e+03 1.499e+03 1.513e+03 1.527e+03 1.541e+03 1.555e+03
 1.569e+03 1.583e+03 1.597e+03 1.611e+03 1.625e+03 1.639e+03 1.653e+03
 1.667e+03 1.681e+03 1.695e+03 1.709e+03 1.723e+03 1.737e+03 1.751e+03
 1.765e+03 1.779e+03 1.793e+03 1.807e+03 1.821e+03 1.835e+03 1.849e+03
 1.863e+03 1.877e+03 1.891e+03 1.905e+03 1.919e+03 1.933e+03 1.947e+03
 1.961e+03 1.975e+03 1.989e+03 2.003e+03 2.017e+03 2.031e+03 2.045e+03
 2.059e+03 2.073e+03 2.087e+03 2.101e+03 2.115e+03 2.129e+03 2.143e+03
 2.157e+03 2.171e+03 2.185e+03 2.199e+03 2.213e+03 2.227e+03 2.241e+03
 2.255e+03 2.269e+03 2.283e+03 2.297e+03 2.311e+03 2.325e+03 2.339e+03
 2.353e+03 2.367e+03 2.381e+03 2.395e+03 2.409e+03 2.423e+03 2.437e+03
 2.451e+03 2.465e+03 2.479e+03 2.493e+03 2.507e+03 2.521e+03 2.535e+03
 2.549e+03 2.563e+03 2.577e+03 2.591e+03 2.605e+03 2.619e+03 2.633e+03
 2.647e+03 2.661e+03 2.675e+03 2.689e+03 2.703e+03 2.717e+03 2.731e+03
 2.745e+03 2.759e+03 2.773e+03 2.787e+03 2.801e+03 2.815e+03 2.829e+03
 2.843e+03 2.857e+03 2.871e+03 2.885e+03 2.899e+03 2.913e+03 2.927e+03
 2.941e+03 2.955e+03 2.969e+03 2.983e+03 2.997e+03 3.011e+03 3.025e+03
 3.039e+03 3.053e+03 3.067e+03 3.081e+03 3.095e+03 3.109e+03 3.123e+03
 3.137e+03 3.151e+03 3.165e+03 3.179e+03 3.193e+03 3.207e+03 3.221e+03
 3.235e+03 3.249e+03 3.263e+03 3.277e+03 3.291e+03 3.305e+03 3.319e+03
 3.333e+03 3.347e+03 3.361e+03 3.375e+03 3.389e+03 3.403e+03 3.417e+03
 3.431e+03 3.445e+03 3.459e+03 3.473e+03 3.487e+03 3.501e+03 3.515e+03
 3.529e+03 3.543e+03 3.557e+03 3.571e+03 3.585e+03 3.599e+03 3.613e+03
 3.627e+03 3.641e+03 3.655e+03 3.669e+03 3.683e+03 3.697e+03 3.711e+03
 3.725e+03 3.739e+03 3.753e+03 3.767e+03 3.781e+03 3.795e+03 3.809e+03
 3.823e+03 3.837e+03 3.851e+03 3.865e+03 3.879e+03 3.893e+03 3.907e+03
 3.921e+03 3.935e+03 3.949e+03 3.963e+03 3.977e+03 3.991e+03 4.005e+03
 4.019e+03 4.033e+03 4.047e+03 4.061e+03 4.075e+03 4.089e+03 4.103e+03
 4.117e+03 4.131e+03 4.145e+03 4.159e+03 4.173e+03 4.187e+03 4.201e+03
 4.215e+03 4.229e+03 4.243e+03 4.257e+03 4.271e+03 4.285e+03 4.299e+03
 4.313e+03 4.327e+03 4.341e+03 4.355e+03 4.369e+03 4.383e+03 4.397e+03
 4.411e+03 4.425e+03 4.439e+03 4.453e+03 4.467e+03 4.481e+03 4.495e+03
 4.509e+03 4.523e+03 4.537e+03 4.551e+03 4.565e+03 4.579e+03 4.593e+03
 4.607e+03 4.621e+03 4.635e+03 4.649e+03 4.663e+03 4.677e+03 4.691e+03
 4.705e+03 4.719e+03 4.733e+03 4.747e+03 4.761e+03 4.775e+03 4.789e+03
 4.803e+03 4.817e+03 4.831e+03 4.845e+03 4.859e+03 4.873e+03 4.887e+03
 4.901e+03 4.915e+03 4.929e+03 4.943e+03 4.957e+03 4.971e+03 4.985e+03
 4.999e+03 5.013e+03 5.027e+03 5.041e+03 5.055e+03 5.069e+03 5.083e+03
 5.097e+03 5.111e+03 5.125e+03 5.139e+03 5.153e+03 5.167e+03 5.181e+03
 5.195e+03 5.209e+03 5.223e+03 5.237e+03 5.251e+03 5.265e+03 5.279e+03
 5.293e+03 5.307e+03 5.321e+03 5.335e+03 5.349e+03 5.363e+03 5.377e+03
 5.391e+03 5.405e+03 5.419e+03 5.433e+03 5.447e+03 5.461e+03 5.475e+03
 5.489e+03 5.503e+03 5.517e+03 5.531e+03 5.545e+03 5.559e+03 5.573e+03
 5.587e+03 5.601e+03 5.615e+03 5.629e+03 5.643e+03 5.657e+03 5.671e+03
 5.685e+03 5.699e+03 5.713e+03 5.727e+03 5.741e+03 5.755e+03 5.769e+03
 5.783e+03 5.797e+03 5.811e+03 5.825e+03 5.839e+03 5.853e+03 5.867e+03
 5.881e+03 5.895e+03 5.909e+03 5.923e+03 5.937e+03 5.951e+03 5.965e+03
 5.979e+03 5.993e+03 6.007e+03 6.021e+03 6.035e+03 6.049e+03 6.063e+03
 6.077e+03 6.091e+03 6.105e+03 6.119e+03 6.133e+03 6.147e+03 6.161e+03
 6.175e+03 6.189e+03 6.203e+03 6.217e+03 6.231e+03 6.245e+03 6.259e+03
 6.273e+03 6.287e+03 6.301e+03 6.315e+03 6.329e+03 6.343e+03 6.357e+03
 6.371e+03 6.385e+03 6.399e+03 6.413e+03 6.427e+03 6.441e+03 6.455e+03
 6.469e+03 6.483e+03 6.497e+03 6.511e+03 6.525e+03 6.539e+03 6.553e+03
 6.567e+03 6.581e+03 6.595e+03 6.609e+03 6.623e+03 6.637e+03 6.651e+03
 6.665e+03 6.679e+03 6.693e+03 6.707e+03 6.721e+03 6.735e+03 6.749e+03
 6.763e+03 6.777e+03 6.791e+03 6.805e+03 6.819e+03 6.833e+03 6.847e+03
 6.861e+03 6.875e+03 6.889e+03 6.903e+03 6.917e+03 6.931e+03]
```

Fig. 4: Quadratic interpolation of I_2 plotted as a function of f_2.



5.3 Basic measurement

- Assign common-mode voltage V_{cm}

In [ ]:
```
Vcm_bm = 0.9
```

- Set bias current I_b (Hint: linear range $I \leq 10$ nA)

In [ ]:
```
p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.NDP_VB_N, \
    pyplane.Coach.BiasType.N, \
    pyplane.Coach.BiasGenMasterCurrent.I30nA, 50)])
```

The bias current is set to

$$I_b = w\frac{BG_{\text{fine}}}{256}I_{BG_{\text{master}}} = \frac{50}{256}\cdot 30\text{nA} \approx 5.859\text{nA}.$$

- Data aquisition

- You can follow the example below

In [ ]:
```
import numpy as np
import time

# your code

V1_Vcm_bm = np.arange(0.75,1.05,0.005) # V1 sweep range

V2_Vcm_bm = []
V1_Vcm_bm_set = []
V2_Vcm_bm_set = []
c2f_Vcm_I1_bm = []
c2f_Vcm_I2_bm = []

for n in range(len(V1_Vcm_bm)):

    # calculate V2 via Vcm and V1
    V2_Vcm_bm.append(2*Vcm_bm-V1_Vcm_bm[n])

    # set V1 and V2
    p.set_voltage(pyplane.DacChannel.AIN5,V1_Vcm_bm[n]) # V1
    p.set_voltage(pyplane.DacChannel.AIN6,V2_Vcm_bm[n]) # V2

    time.sleep(0.5) # settle time

    # get set V1 and V2
    V1_Vcm_bm_set.append(p.get_set_voltage(pyplane.DacChannel.AIN5))
    V2_Vcm_bm_set.append(p.get_set_voltage(pyplane.DacChannel.AIN6))

    # read c2f values
    c2f_Vcm_temp = p.read_c2f_output(0.1)
    c2f_Vcm_I1_bm.append(c2f_Vcm_temp[0])
    c2f_Vcm_I2_bm.append(c2f_Vcm_temp[1])

print(V1_Vcm_bm)
print(V2_Vcm_bm)
print(c2f_Vcm_I1_bm)
print(c2f_Vcm_I2_bm)

data_Vcm_bm = [V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I
np.savetxt('c2f_Vcm_bm_vs_V1_V2.csv', data_Vcm_bm, delimiter=',')
```

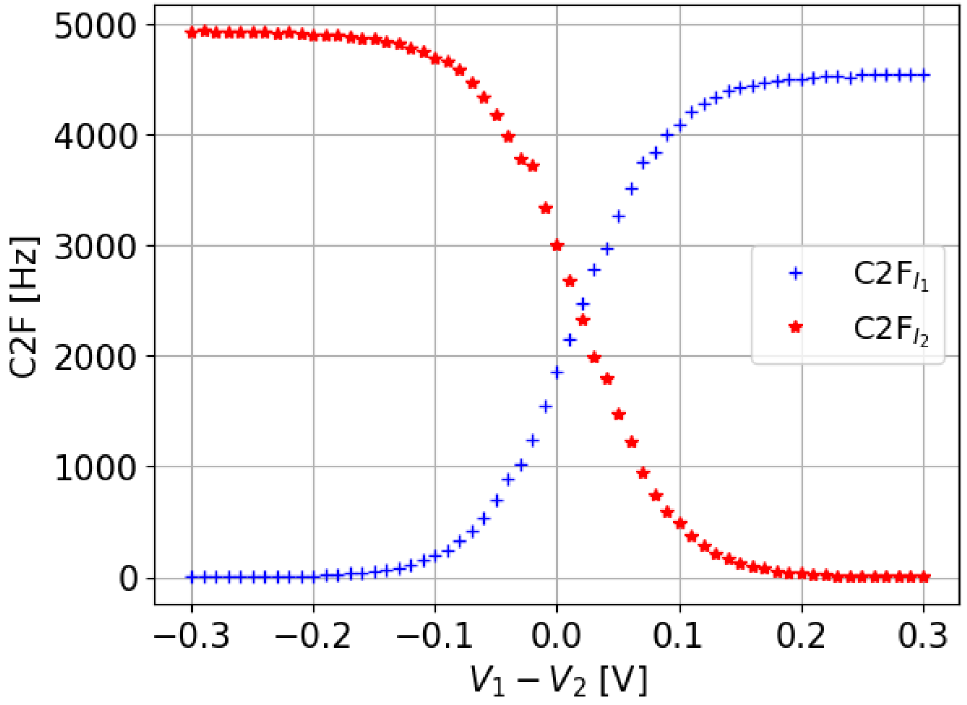
```
[0.75  0.755 0.76  0.765 0.77  0.775 0.78  0.785 0.79  0.795 0.8   0.805
 0.81  0.815 0.82  0.825 0.83  0.835 0.84  0.845 0.85  0.855 0.86  0.865
 0.87  0.875 0.88  0.885 0.89  0.895 0.9   0.905 0.91  0.915 0.92  0.925
 0.93  0.935 0.94  0.945 0.95  0.955 0.96  0.965 0.97  0.975 0.98  0.985
 0.99  0.995 1.    1.005 1.01  1.015 1.02  1.025 1.03  1.035 1.04  1.045
 1.05 ]
[1.05, 1.045, 1.04, 1.0350000000000001, 1.03, 1.025, 1.02, 1.0150000000000001, 1.01,
1.005, 1.0, 0.995, 0.99, 0.985, 0.98, 0.975, 0.97, 0.965, 0.96, 0.955, 0.95, 0.945,
0.94, 0.9349999999999999, 0.9299999999999999, 0.9249999999999999, 0.9199999999999999,
0.9149999999999999, 0.9099999999999999, 0.9049999999999999, 0.8999999999999999, 0.894
9999999999999, 0.8899999999999999, 0.8849999999999999, 0.8799999999999999, 0.87499999
99999999, 0.8699999999999999, 0.8649999999999999, 0.8599999999999999, 0.8549999999999
999, 0.8499999999999999, 0.8449999999999999, 0.8399999999999999, 0.8349999999999999,
0.8299999999999998, 0.8249999999999998, 0.8199999999999998, 0.8149999999999998, 0.809
9999999999998, 0.8049999999999998, 0.7999999999999998, 0.7949999999999997, 0.78999999
99999998, 0.7849999999999999, 0.7799999999999998, 0.7749999999999997, 0.7699999999999
998, 0.7649999999999999, 0.7599999999999998, 0.7549999999999997, 0.7499999999999998]
[3, 3, 3, 4, 4, 5, 6, 7, 8, 11, 13, 18, 23, 30, 40, 49, 65, 87, 115, 155, 204, 249, 3
24, 415, 534, 701, 894, 1027, 1246, 1547, 1863, 2149, 2481, 2783, 2978, 3266, 3521, 3
751, 3836, 4009, 4089, 4201, 4281, 4343, 4394, 4426, 4436, 4472, 4493, 4497, 4500, 45
14, 4533, 4527, 4522, 4538, 4545, 4551, 4541, 4545, 4548]
[4933, 4937, 4930, 4923, 4921, 4922, 4921, 4915, 4923, 4913, 4899, 4900, 4900, 4883,
4867, 4869, 4844, 4828, 4786, 4755, 4688, 4667, 4582, 4476, 4341, 4176, 3989, 3783, 3
727, 3338, 3010, 2675, 2328, 1983, 1802, 1478, 1221, 941, 747, 589, 493, 375, 289, 21
6, 164, 122, 100, 74, 56, 41, 31, 23, 18, 15, 11, 9, 7, 6, 5, 4, 4]
```

- Plot raw data (frequency)
- You can follow the example below (but remember to save data firstly)

In [ ]:

```python
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})

V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I2_bm = np.loadt

range_V1V2_bm = V1_Vcm_bm - V2_Vcm_bm

plt.plot(range_V1V2_bm,c2f_Vcm_I1_bm,'b+',range_V1V2_bm,c2f_Vcm_I2_bm,'r*')

plt.xlabel('$V_1-V_2$ [V]')
plt.ylabel('C2F [Hz]')
plt.legend(['C2F$_{I_1}$','C2F$_{I_2}$'],prop={'size': 14})
plt.title('Fig. 5: Measured C2F data for $I_1$ and $I_2$ plotted over $V_1-V_2$.')
plt.grid()
plt.show()
```

Fig. 5: Measured C2F data for I_1 and I_2 plotted over $V_1 - V_2$.



- Save raw data
- You can follow the example below

```
In [ ]: # if the data looks nice, save it!
        data_Vcm_bm = [V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I
        # save to csv file
        np.savetxt('c2f_Vcm_bm_vs_V1_V2.csv', data_Vcm_bm, delimiter=',')
```

- Convert frequency to current
- You can follow the example below

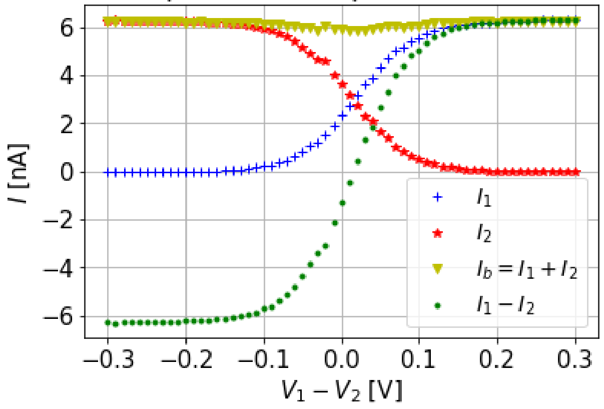
```
In [ ]: # Use bias measurements
        V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I2_bm = np.loadt

        I1_bm = a0_I1cal+a1_I1cal*np.array(c2f_Vcm_I1_bm)+a2_I1cal*np.array(c2f_Vcm_I1_bm)**2
        I2_bm = a0_I2cal+a1_I2cal*np.array(c2f_Vcm_I2_bm)+a2_I2cal*np.array(c2f_Vcm_I2_bm)**2

        #for later
        I1_bm_f6 = I1_bm
        I2_bm_f6 = I2_bm
```

- Plot I_1, I_2, $I_1 + I_2$, $I_1 - I_2$
- You can follow the example below

```
In [ ]: import matplotlib.pyplot as plt
        plt.rcParams.update({'font.size': 15})

        range_V1V2_bm = V1_Vcm_bm - V2_Vcm_bm
```

```
plt.plot(range_V1V2_bm,I1_bm,'b+')
plt.plot(range_V1V2_bm,I2_bm,'r*')
plt.plot(range_V1V2_bm,I1_bm+I2_bm,'yv')
plt.plot(range_V1V2_bm,I1_bm-I2_bm,'g.')

plt.xlabel('$V_1-V_2$ [V]')
plt.ylabel('$I$ [nA]')
plt.legend(['$I_1$','$I_2$','$I_b=I_1+I_2$','$I_1-I_2$'],prop={'size': 14})
plt.title('Fig. 6: Interpolated differential pair currents plotted over the voltage di
plt.grid()
plt.show()
```

Fig. 6: Interpolated differential pair currents plotted over the voltage difference $V_1 - V_2$.



5.4 Bias variation

Repeat the measurement for a different value of I_b

- Use the same common-mode voltage V_{cm} as in 5.3

In [ ]:
```
Vcm_bv = 0.9
```

- Set the new bias current (Hint: linear range $I \leq 10$ nA)

In [ ]:
```
p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.NDP_VB_N, \
    pyplane.Coach.BiasType.N, \
    pyplane.Coach.BiasGenMasterCurrent.I30nA, 20)])
```

The bias current was changed from $I_b \approx 5.859\text{nA}$ to $I_b = \frac{20}{256} \cdot 30\text{nA} \approx 2.344\text{nA}$.

- Data aquisition

In [ ]:
```
import numpy as np
import time

# your code

V1_Vcm_bm = np.arange(0.75,1.05,0.005) # V1 sweep range
```

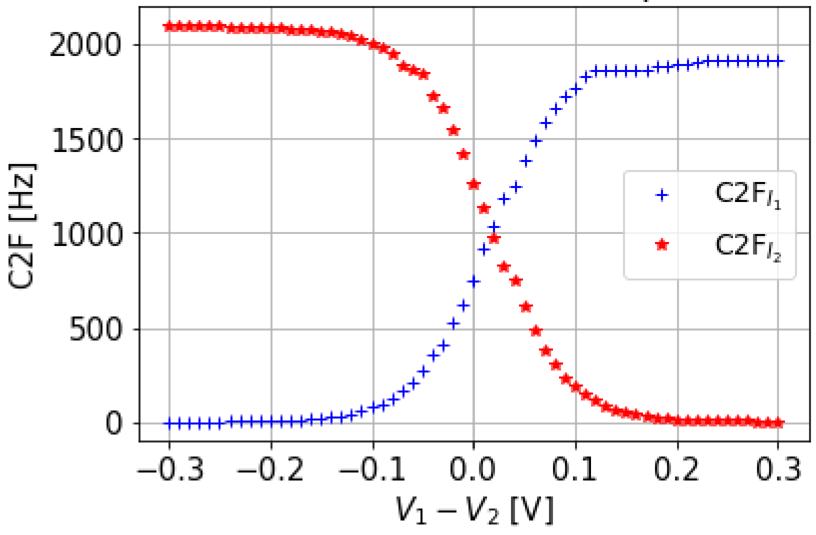
```
V2_Vcm_bm = []
V1_Vcm_bm_set = []
V2_Vcm_bm_set = []
c2f_Vcm_I1_bm = []
c2f_Vcm_I2_bm = []

for n in range(len(V1_Vcm_bm)):

    # calculate V2 via Vcm and V1
    V2_Vcm_bm.append(2*Vcm_bm-V1_Vcm_bm[n])

    # set V1 and V2
    p.set_voltage(pyplane.DacChannel.AIN5,V1_Vcm_bm[n]) # V1
    p.set_voltage(pyplane.DacChannel.AIN6,V2_Vcm_bm[n]) # V2

    time.sleep(0.5) # settle time

    # get set V1 and V2
    V1_Vcm_bm_set.append(p.get_set_voltage(pyplane.DacChannel.AIN5))
    V2_Vcm_bm_set.append(p.get_set_voltage(pyplane.DacChannel.AIN6))

    # read c2f values
    c2f_Vcm_temp = p.read_c2f_output(0.1)
    c2f_Vcm_I1_bm.append(c2f_Vcm_temp[0])
    c2f_Vcm_I2_bm.append(c2f_Vcm_temp[1])

print(V1_Vcm_bm)
print(V2_Vcm_bm)
print(c2f_Vcm_I1_bm)
print(c2f_Vcm_I2_bm)

data_Vcm_bm = [V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I
np.savetxt('new_biais_c2f_Vcm_bm_vs_V1_V2.csv', data_Vcm_bm, delimiter=',')
```

```
[0.75  0.755 0.76  0.765 0.77  0.775 0.78  0.785 0.79  0.795 0.8   0.805
 0.81  0.815 0.82  0.825 0.83  0.835 0.84  0.845 0.85  0.855 0.86  0.865
 0.87  0.875 0.88  0.885 0.89  0.895 0.9   0.905 0.91  0.915 0.92  0.925
 0.93  0.935 0.94  0.945 0.95  0.955 0.96  0.965 0.97  0.975 0.98  0.985
 0.99  0.995 1.    1.005 1.01  1.015 1.02  1.025 1.03  1.035 1.04  1.045
 1.05 ]
[1.05, 1.045, 1.04, 1.0350000000000001, 1.03, 1.025, 1.02, 1.0150000000000001, 1.01,
1.005, 1.0, 0.995, 0.99, 0.985, 0.98, 0.975, 0.97, 0.965, 0.96, 0.955, 0.95, 0.945,
0.94, 0.9349999999999999, 0.9299999999999999, 0.9249999999999999, 0.9199999999999999,
0.9149999999999999, 0.9099999999999999, 0.9049999999999999, 0.8999999999999999, 0.894
9999999999999, 0.8899999999999999, 0.8849999999999999, 0.8799999999999999, 0.87499999
99999999, 0.8699999999999999, 0.8649999999999999, 0.8599999999999999, 0.8549999999999
999, 0.8499999999999999, 0.8449999999999999, 0.8399999999999999, 0.8349999999999999,
0.8299999999999998, 0.8249999999999998, 0.8199999999999998, 0.8149999999999998, 0.809
9999999999998, 0.8049999999999998, 0.7999999999999998, 0.7949999999999997, 0.78999999
99999998, 0.7849999999999999, 0.7799999999999998, 0.7749999999999997, 0.7699999999999
998, 0.7649999999999999, 0.7599999999999998, 0.7549999999999997, 0.7499999999999998]
[3, 3, 3, 3, 3, 3, 4, 4, 5, 5, 6, 8, 10, 13, 16, 20, 26, 34, 45, 59, 80, 95, 126, 16
4, 215, 278, 356, 415, 525, 622, 749, 916, 1039, 1183, 1244, 1388, 1494, 1587, 1666,
1731, 1769, 1835, 1866, 1864, 1864, 1865, 1865, 1866, 1880, 1881, 1895, 1899, 1906, 1
912, 1914, 1919, 1914, 1917, 1920, 1920, 1919]
[2096, 2094, 2096, 2099, 2095, 2098, 2089, 2091, 2086, 2084, 2089, 2086, 2079, 2076,
2073, 2070, 2066, 2052, 2041, 2027, 2002, 1982, 1949, 1881, 1864, 1838, 1724, 1660, 1
547, 1423, 1260, 1137, 975, 830, 748, 612, 489, 384, 301, 234, 194, 146, 111, 83, 62,
47, 38, 29, 22, 17, 13, 10, 8, 7, 6, 5, 4, 4, 3, 3, 3]
```

- Plot raw data (frequency)

```
In [ ]: import matplotlib.pyplot as plt
        plt.rcParams.update({'font.size': 15})

        V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I2_bm = np.loadt

        range_V1V2_bm = V1_Vcm_bm - V2_Vcm_bm

        plt.plot(range_V1V2_bm,c2f_Vcm_I1_bm,'b+',range_V1V2_bm,c2f_Vcm_I2_bm,'r*')

        plt.xlabel('$V_1-V_2$ [V]')
        plt.ylabel('C2F [Hz]')
        plt.legend(['C2F$_{I_1}$','C2F$_{I_2}$'],prop={'size': 14})
        plt.title('Fig. 7: Measured C2F data for $I_1$ and $I_2$ plotted over $V_1-V_2$.')
        plt.grid()
        plt.show()
```

Fig. 7: Measured C2F data for I_1 and I_2 plotted over $V_1 - V_2$.



- Save raw data

Done previously

- Convert frequency to current

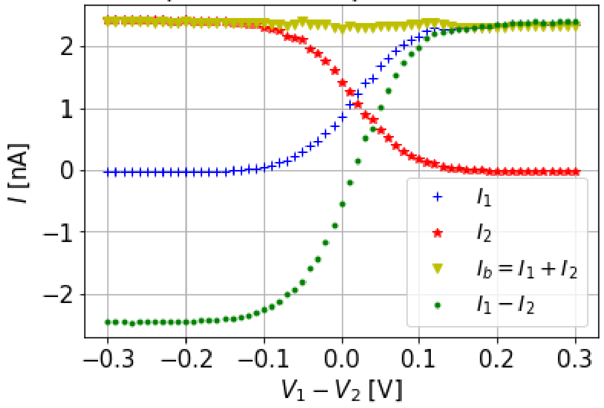
```
In [ ]: # Use bias measurements
        V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I2_bm = np.loadt

        I1_bm = a0_I1cal+a1_I1cal*np.array(c2f_Vcm_I1_bm)+a2_I2cal*np.array(c2f_Vcm_I1_bm)**2
        I2_bm = a0_I2cal+a1_I2cal*np.array(c2f_Vcm_I2_bm)+a2_I2cal*np.array(c2f_Vcm_I2_bm)**2

        #for later
        I1_bm_f8 = I1_bm
        I2_bm_f8 = I2_bm
```

The C2F data can now be mapped to the corresponding currents I_1 and I_2 using the determined quadratic interpolation functions $I_1(f_1)$ and $I_2(f_2)$.

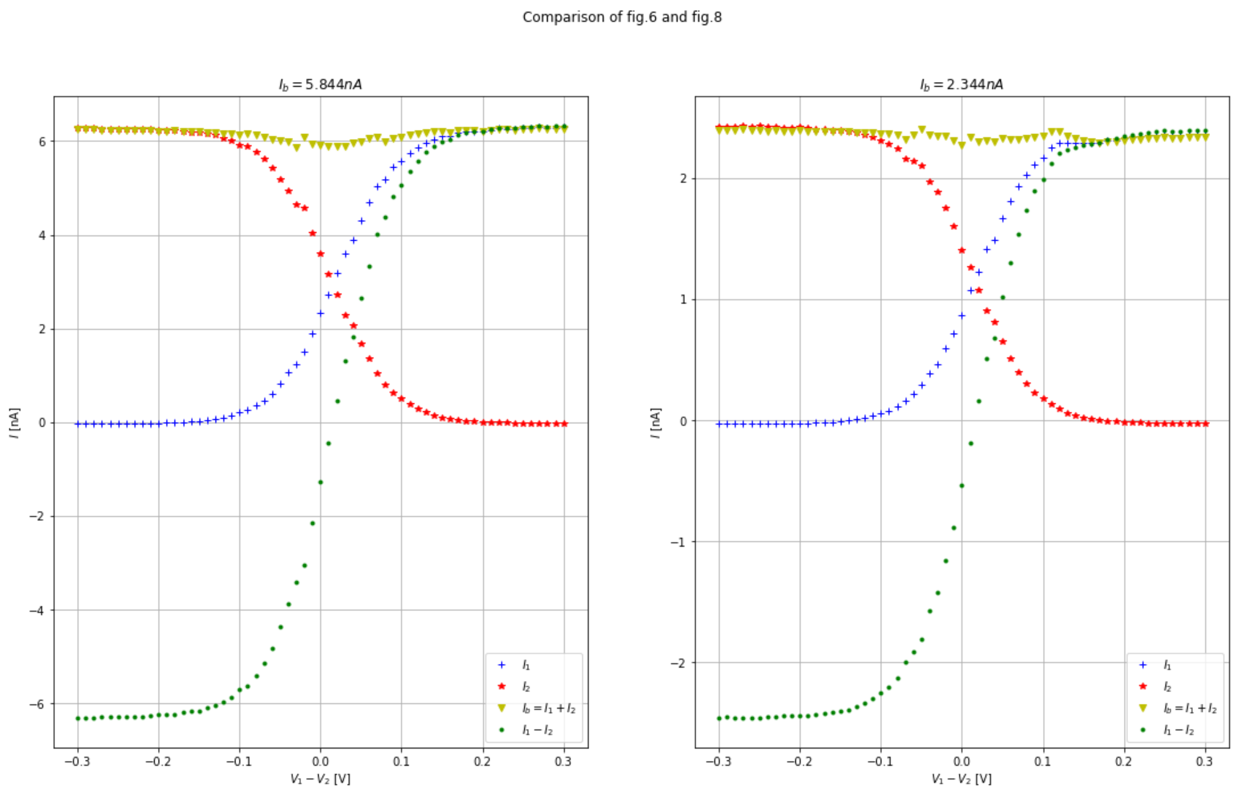
- Plot I_1, I_2, $I_1 + I_2$, $I_1 - I_2$ and compare it with Fig. 6.

In [ ]:
```
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})

range_V1V2_bm = V1_Vcm_bm - V2_Vcm_bm


plt.plot(range_V1V2_bm,I1_bm,'b+')
plt.plot(range_V1V2_bm,I2_bm,'r*')
plt.plot(range_V1V2_bm,I1_bm+I2_bm,'yv')
plt.plot(range_V1V2_bm,I1_bm-I2_bm,'g.')

plt.xlabel('$V_1-V_2$ [V]')
plt.ylabel('$I$ [nA]')
plt.legend(['$I_1$','$I_2$','$I_b=I_1+I_2$','$I_1-I_2$'],prop={'size': 14})
plt.title('Fig. 8: Interpolated differential pair currents plotted over the voltage di
plt.grid()
plt.show()
```

Fig. 8: Interpolated differential pair currents plotted over the voltage difference $V_1 - V_2$.



In [ ]:
```
fig, (ax6, ax8) = plt.subplots(1, 2)
fig.set_size_inches(18.5, 10.5)
fig.suptitle('Comparison of fig.6 and fig.8')
ax6.plot(range_V1V2_bm,I1_bm_f6,'b+')
ax6.plot(range_V1V2_bm,I2_bm_f6,'r*')
ax6.plot(range_V1V2_bm,I1_bm_f6+I2_bm_f6,'yv')
ax6.plot(range_V1V2_bm,I1_bm_f6-I2_bm_f6,'g.')

ax6.set(xlabel = '$V_1-V_2$ [V]',ylabel ='$I$ [nA]')
ax6.legend(['$I_1$','$I_2$','$I_b=I_1+I_2$','$I_1-I_2$'],prop={'size': 10})
ax6.set_title('$I_{b}=5.844 nA$')
ax6.grid()


ax8.plot(range_V1V2_bm,I1_bm_f8,'b+')
ax8.plot(range_V1V2_bm,I2_bm_f8,'r*')
ax8.plot(range_V1V2_bm,I1_bm_f8+I2_bm_f8,'yv')
ax8.plot(range_V1V2_bm,I1_bm_f8-I2_bm_f8,'g.')

ax8.set(xlabel = '$V_1-V_2$ [V]',ylabel ='$I$ [nA]')
ax8.legend(['$I_1$','$I_2$','$I_b=I_1+I_2$','$I_1-I_2$'],prop={'size': 10})
ax8.set_title('$I_{b}=2.344 nA$')
ax8.grid()
```



Compared to Fig. 6, Fig. 8 has a much wider amplitude for the output currents. (which makes sense: if I_b is lower I_1 and I_2 will be smaller as well)

5.5 Sensitivity to input common mode

Repeat the measurement for a different value of V_{cm}

- Set a new common-mode voltage V_{cm}

In [ ]:
```
Vcm_cmv = 1.3
Vcm_bm = Vcm_cmv #just not to rename all the variables
```

Common-mode voltage was changed from $V_{cm} = 0.9\text{V}$ to $V_{cm} = 1.3\text{V}$.

- Use the same bias current I_b as 5.3

In [ ]:
```
p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.NDP_VB_N, \
    pyplane.Coach.BiasType.N, \
    pyplane.Coach.BiasGenMasterCurrent.I30nA, 50)])
```

- Data aquisition

In [ ]:
```
import numpy as np
import time
```

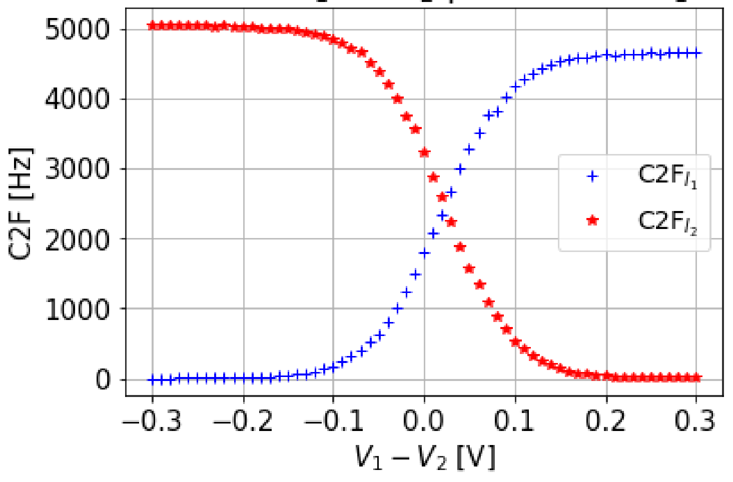
```
# your code

V1_Vcm_bm = np.arange(1.15,1.45,0.005) # V1 sweep range (start at 0.80 or else V2 is r

V2_Vcm_bm = []
V1_Vcm_bm_set = []
V2_Vcm_bm_set = []
c2f_Vcm_I1_bm = []
c2f_Vcm_I2_bm = []

for n in range(len(V1_Vcm_bm)):

    # calculate V2 via Vcm and V1
    V2_Vcm_bm.append(2*Vcm_bm-V1_Vcm_bm[n])

    # set V1 and V2

    p.set_voltage(pyplane.DacChannel.AIN5,V1_Vcm_bm[n]) # V1
    p.set_voltage(pyplane.DacChannel.AIN6,V2_Vcm_bm[n]) # V2

    time.sleep(0.5) # settle time

    # get set V1 and V2
    V1_Vcm_bm_set.append(p.get_set_voltage(pyplane.DacChannel.AIN5))
    V2_Vcm_bm_set.append(p.get_set_voltage(pyplane.DacChannel.AIN6))

    # read c2f values
    c2f_Vcm_temp = p.read_c2f_output(0.1)
    c2f_Vcm_I1_bm.append(c2f_Vcm_temp[0])
    c2f_Vcm_I2_bm.append(c2f_Vcm_temp[1])

print(V1_Vcm_bm)
print(V2_Vcm_bm)
print(c2f_Vcm_I1_bm)
print(c2f_Vcm_I2_bm)

data_Vcm_bm = [V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I
np.savetxt('new_Vcm_c2f_Vcm_bm_vs_V1_V2.csv', data_Vcm_bm, delimiter=',')
```

```
[1.15  1.155 1.16  1.165 1.17  1.175 1.18  1.185 1.19  1.195 1.2   1.205
 1.21  1.215 1.22  1.225 1.23  1.235 1.24  1.245 1.25  1.255 1.26  1.265
 1.27  1.275 1.28  1.285 1.29  1.295 1.3   1.305 1.31  1.315 1.32  1.325
 1.33  1.335 1.34  1.345 1.35  1.355 1.36  1.365 1.37  1.375 1.38  1.385
 1.39  1.395 1.4   1.405 1.41  1.415 1.42  1.425 1.43  1.435 1.44  1.445
 1.45 ]
[1.4500000000000002, 1.4450000000000003, 1.4400000000000004, 1.4350000000000005, 1.43
00000000000006, 1.4250000000000007, 1.4200000000000008, 1.415000000000001, 1.41000000
0000001, 1.4050000000000011, 1.4000000000000012, 1.3950000000000014, 1.39000000000000
15, 1.3850000000000016, 1.3800000000000017, 1.3750000000000018, 1.3700000000000019,
1.365000000000002, 1.360000000000002, 1.3550000000000022, 1.3500000000000023, 1.34500
00000000024, 1.3400000000000025, 1.3350000000000026, 1.330000000000027, 1.3250000000
000028, 1.320000000000003, 1.315000000000003, 1.3100000000000032, 1.3050000000000033,
1.3000000000000034, 1.2950000000000035, 1.2900000000000036, 1.2850000000000037, 1.280
0000000000038, 1.275000000000004, 1.270000000000004, 1.2650000000000041, 1.2600000000
000042, 1.2550000000000043, 1.2500000000000044, 1.2450000000000045, 1.240000000000004
7, 1.2350000000000048, 1.2300000000000049, 1.225000000000005, 1.220000000000005, 1.21
50000000000052, 1.2100000000000053, 1.2050000000000054, 1.2000000000000055, 1.1950000
000000056, 1.1900000000000057, 1.1850000000000058, 1.180000000000006, 1.1750000000000
06, 1.1700000000000061, 1.1650000000000063, 1.1600000000000064, 1.1550000000000065,
1.1500000000000066]
[3, 3, 3, 4, 4, 5, 5, 6, 8, 10, 12, 16, 21, 26, 35, 46, 62, 79, 105, 136, 181, 235, 3
13, 395, 514, 624, 806, 1016, 1247, 1490, 1792, 2082, 2351, 2678, 2993, 3286, 3524, 3
756, 3825, 4028, 4171, 4289, 4360, 4432, 4488, 4522, 4550, 4577, 4596, 4615, 4633, 46
22, 4640, 4643, 4645, 4651, 4646, 4656, 4662, 4657, 4659]
[5049, 5042, 5046, 5033, 5047, 5049, 5035, 5030, 5034, 5007, 5014, 5015, 5005, 4991,
5001, 4992, 4967, 4953, 4917, 4892, 4850, 4791, 4712, 4650, 4516, 4383, 4206, 3987, 3
745, 3564, 3221, 2879, 2585, 2242, 1869, 1586, 1341, 1078, 895, 696, 535, 416, 326, 2
44, 192, 142, 105, 78, 60, 45, 33, 26, 20, 15, 11, 9, 7, 6, 5, 4, 4]
```

- Plot raw data (frequency)

In [ ]:
```python
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})

V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I2_bm = np.loadt

range_V1V2_bm = V1_Vcm_bm - V2_Vcm_bm

plt.plot(range_V1V2_bm,c2f_Vcm_I1_bm,'b+',range_V1V2_bm,c2f_Vcm_I2_bm,'r*')

plt.xlabel('$V_1-V_2$ [V]')
plt.ylabel('C2F [Hz]')
plt.legend(['C2F$_{I_1}$','C2F$_{I_2}$'],prop={'size': 14})
plt.title('Fig. 9: Measured C2F data for $I_1$ and $I_2$ plotted over $V_1-V_2$ with $
plt.grid()
plt.show()
```

Fig. 9: Measured C2F data for I_1 and I_2 plotted over $V_1 - V_2$ with $V_{cm} = 1.3V$



- Save raw data

Already done

- Convert frequency to current

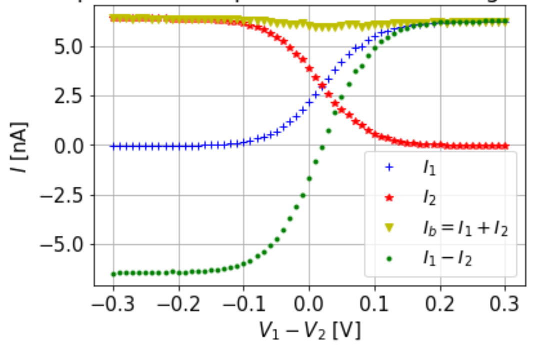
```
In [ ]: # Use bias measurements
V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I2_bm = np.loadt
I1_bm = a0_I1cal+a1_I1cal*np.array(c2f_Vcm_I1_bm)+a2_I2cal*np.array(c2f_Vcm_I1_bm)**2
I2_bm = a0_I2cal+a1_I2cal*np.array(c2f_Vcm_I2_bm)+a2_I2cal*np.array(c2f_Vcm_I2_bm)**2

#for later
I1_bm_f10 = I1_bm
I2_bm_f10 = I2_bm
```

The C2F data can now be mapped to the corresponding currents I_1 and I_2 using the determined quadratic interpolation functions $I_1(f_1)$ and $I_2(f_2)$.

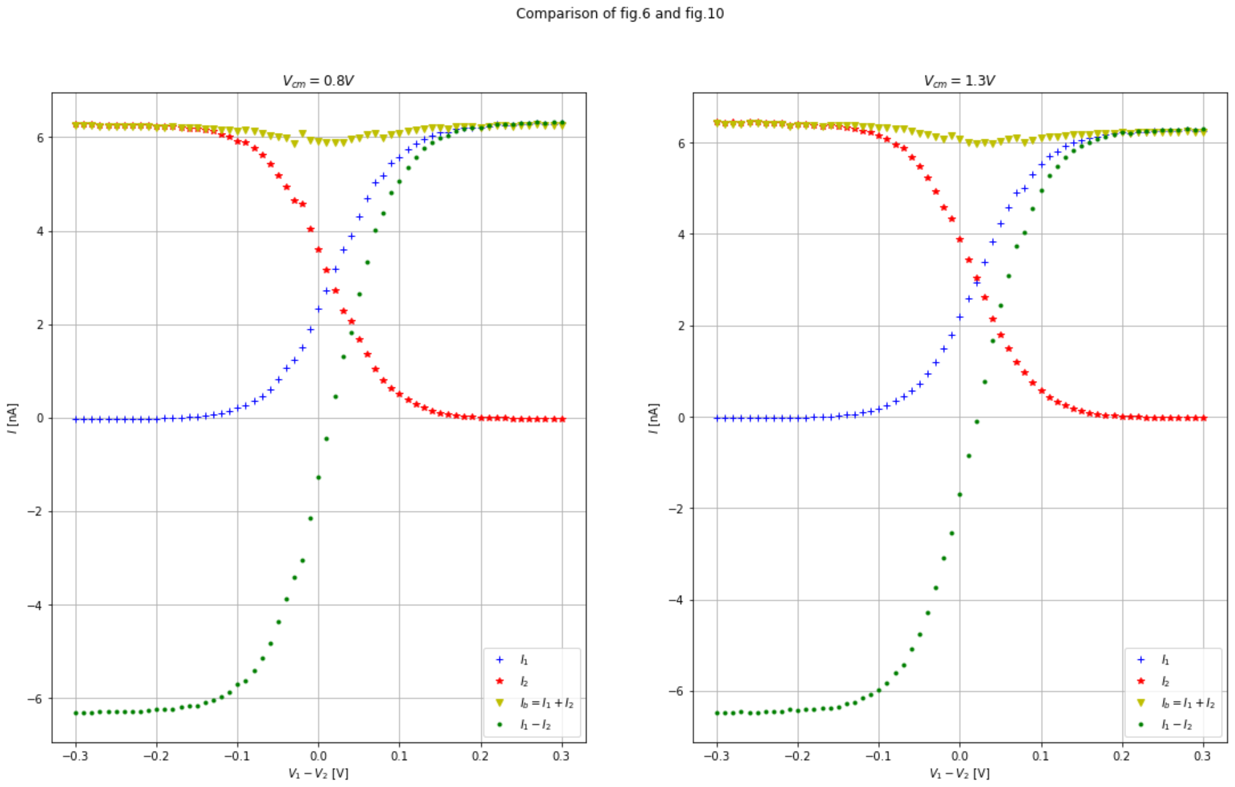
- Plot I_1, I_2, $I_1 + I_2$, $I_1 - I_2$ and compare it with Fig. 6.

```
In [ ]: import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})

range_V1V2_bm = V1_Vcm_bm - V2_Vcm_bm


plt.plot(range_V1V2_bm,I1_bm,'b+')
plt.plot(range_V1V2_bm,I2_bm,'r*')
plt.plot(range_V1V2_bm,I1_bm+I2_bm,'yv')
plt.plot(range_V1V2_bm,I1_bm-I2_bm,'g.')

plt.xlabel('$V_1-V_2$ [V]')
plt.ylabel('$I$ [nA]')
plt.legend(['$I_1$','$I_2$','$I_b=I_1+I_2$','$I_1-I_2$'],prop={'size': 14})
plt.title('Fig. 10: Interpolated differential pair currents plotted over the voltage d
plt.grid()
plt.show()
```

Fig. 10: Interpolated differential pair currents plotted over the voltage difference $V_1 - V_2$ with $V_{cm} = 1.3V$



```
In [ ]: fig, (ax6, ax10) = plt.subplots(1, 2)
        fig.set_size_inches(18.5, 10.5)
        fig.suptitle('Comparison of fig.6 and fig.10')
        ax6.plot(range_V1V2_bm,I1_bm_f6,'b+')
        ax6.plot(range_V1V2_bm,I2_bm_f6,'r*')
        ax6.plot(range_V1V2_bm,I1_bm_f6+I2_bm_f6,'yv')
        ax6.plot(range_V1V2_bm,I1_bm_f6-I2_bm_f6,'g.')

        ax6.set(xlabel = '$V_1-V_2$ [V]',ylabel ='$I$ [nA]')
        ax6.legend(['$I_1$','$I_2$','$I_b=I_1+I_2$','$I_1-I_2$'],prop={'size': 10})
        ax6.set_title('$V_{cm}=0.8 V$')
        ax6.grid()


        ax10.plot(range_V1V2_bm,I1_bm_f10,'b+')
        ax10.plot(range_V1V2_bm,I2_bm_f10,'r*')
        ax10.plot(range_V1V2_bm,I1_bm_f10+I2_bm_f10,'yv')
        ax10.plot(range_V1V2_bm,I1_bm_f10-I2_bm_f10,'g.')

        ax10.set(xlabel = '$V_1-V_2$ [V]',ylabel ='$I$ [nA]')
        ax10.legend(['$I_1$','$I_2$','$I_b=I_1+I_2$','$I_1-I_2$'],prop={'size': 10})
        ax10.set_title('$V_{cm}=1.3 V$')
        ax10.grid()
```



At first sight we notice no significant difference between the two figures

5.6 Analysis

- Comment on the range of linearity and on the measured offset voltage (the voltage that makes $I_1 = I_2$).

The range of linearity occurs approximately in between $V_1 - V_2 \in [-0.1, 0.1]$.

This makes sense considering the current equations. Also, $I_1 - I_2$ can be rewritten as a $\tanh()$ function (and also has the same range of linearity as I_1 and I_2):

$$I_1 = I_b \frac{e^{\frac{\kappa v_1}{U_T}}}{e^{\frac{\kappa V_1}{U_T}} + e^{\frac{\kappa V_2}{U_T}}}$$

$$I_2 = I_b \frac{e^{\frac{\kappa V_2}{U_T}}}{e^{\frac{\kappa V_1}{U_T}} + e^{\frac{\kappa V_2}{U_T}}}$$

For the offset voltage: When $V_1 = V_2$,then $I_1 = I_2 = \frac{I_b}{2}$. (We get this by plugging $V_1 = V_2$ in the previous equations)

- What determines the linear range of input voltage?

The biais current determines the slope of the linear range of the input voltage (proportionally) but the range itself is influenced by kappa (and the thermal voltage)

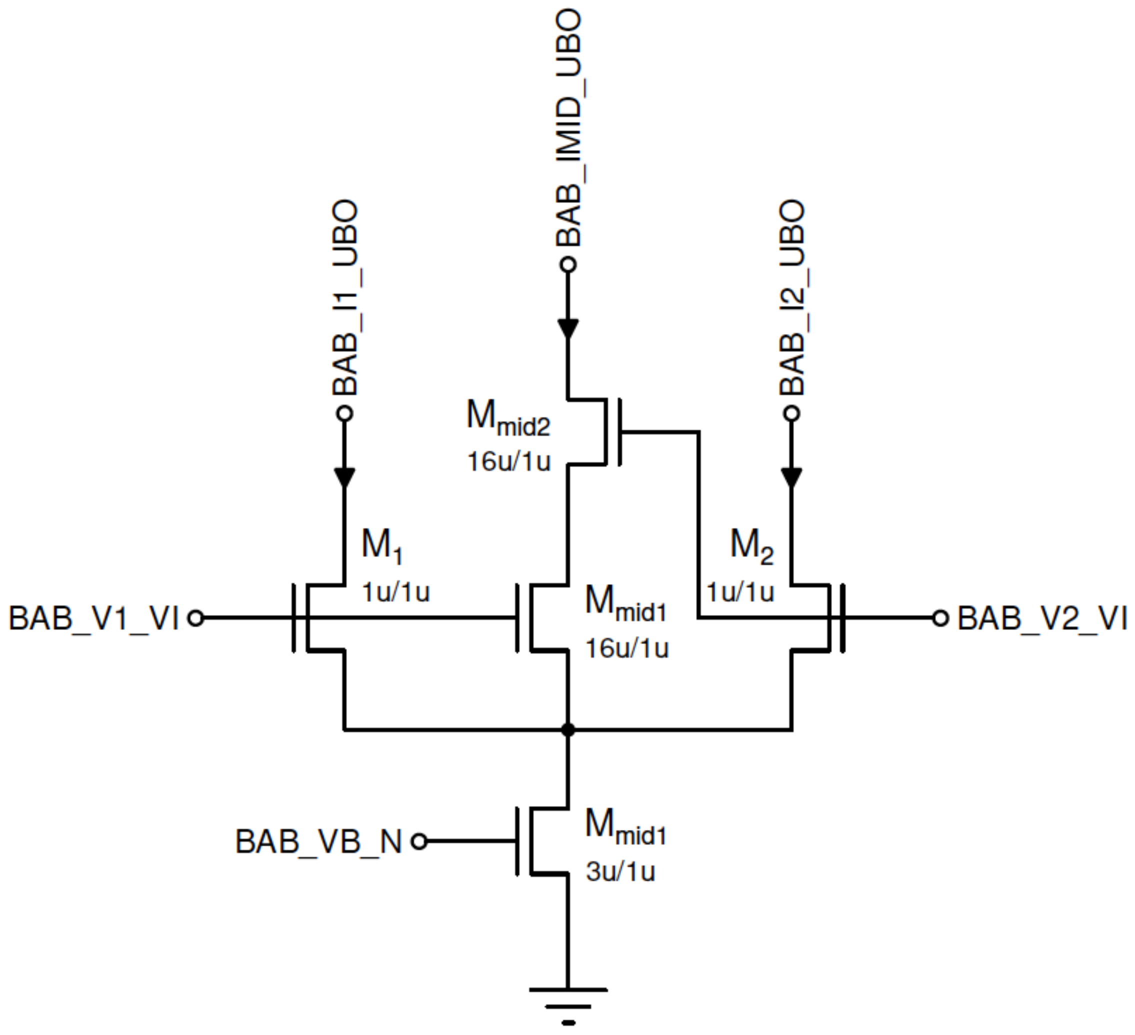
- If you were to run the differential pair in strong inversion, what voltage would determine the linear range of operation? Hint: In weak inversion the thermal voltage is the natural voltage scale. In strong inversion, what is the most natural voltage scale?

Not covered in the course yet

6 Bump-antibump circuit (BAB)

In this experiment, we will measure the input-output relationship of the bump-antibump circuit.

6.0 Schematic and pin map



I_1 **= BAB_I1_UBO = C2F[5]**

I_2 **= BAB_I2_UBO = C2F[6]**

I_{out} = **BAB_IMID_UBO = C2F[7]**

V_1 = **BAB_V1_VI = AIN12**

V_2 = **BAB_V2_VI = AIN13**

6.1 Chip configuration

In [ ]:
```
p.send_coach_events([pyplane.Coach.generate_aerc_event( \
    pyplane.Coach.CurrentOutputSelect.SelectLine5, \
    pyplane.Coach.VoltageOutputSelect.NoneSelected, \
    pyplane.Coach.VoltageInputSelect.SelectLine2, \
    pyplane.Coach.SynapseSelect.NoneSelected, 0)])
```

Assume the W/L ratio between the bump-antibump bias transistor Mb and the BiasGen output transistor is **3**.

- If we trust the value for I_b calculated from the BiasGen, how do we find out the mapping between I and f for each C2F channel?

From the schematic of the bump-antibump circuit, it can be inferred that (KCL)

$$I_b = I_1 + I_2 + I_{out}.$$

As the current I_1 becomes far larger than the currents I_2 and I_{out} for $V_1 \gg V_2$ (and the current I_2 becomes far larger than the currents I_1 and I_{out} for $V_1 \ll V_2$), the following approximations can be made

$$V_1 \gg V_2: \quad I_1 \approx I_b \Rightarrow f_1 \approx \frac{I_b}{C\Delta U} \quad \text{and}$$

$$V_1 \ll V_2: \quad I_2 \approx I_b \Rightarrow f_2 \approx \frac{I_b}{C\Delta U}.$$

this property can be utilized to find the mapping between I_1 and $f_1(I_b)$ and I_2 and $f_2(I_b)$.

And analogous procedure can not be performed for I_{out}, as this would require both I_1 and I_2 to be 0. In this case, V_1 and V_2 would have to be 0 as well, implying that all gates in the circuit are closed and no currents are flowing.

However, in the prelab it was determined that

$$I_{out} = I_b \frac{r}{r + 4\cosh^2\left(\frac{\kappa \Delta V}{2U_T}\right)}.$$

Thus, it can be inferred that for $\Delta V = 0 \Rightarrow \cosh^2\left(\frac{\kappa \Delta V}{2U_T}\right) = 1$

$$I_{out} = I_b \frac{r}{r+4},$$

where r is the W/L ratio between M_{mid1} and M_1 or M_{mid2} and M_2 respectively

$$r = \frac{16\mu\text{m}}{1\mu\text{m}} = 16.$$

Thus

$I_{out} = I_b \frac{16}{16+4} = \frac{4}{5} I_b$, yielding the condition

$$V_1, V_2 \neq 0 : V_1 - V_2 = 0 : \quad \frac{4}{5} I_b \approx I_1 \Rightarrow f_{out} \approx \frac{4}{5} \frac{I_b}{C \Delta U} \quad .$$

6.2 C2F calibration

6.2.1 Calibrate C2F response for I1

- Set fixed voltages for V_1 and V_2

```
In [ ]: p.set_voltage(pyplane.DacChannel.AIN12,0.7) # V1 = 0.7
        p.set_voltage(pyplane.DacChannel.AIN13,0.2) # V2 = 0.2
```

```
Out[ ]: 0.19882699847221375
```

Set V_1 and V_2 such that $V_1 \gg V_2$.

- Data aquisition (Hint: linear range $I \leq 10$ nA)

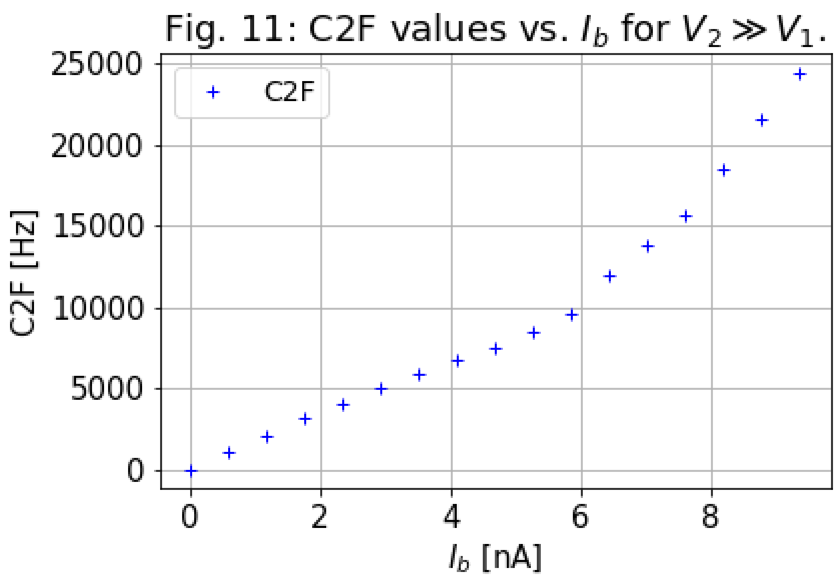
```
In [ ]: import pyplane
        import numpy as np
        import time
        import matplotlib.pyplot as plt
        # your code

        bg_fine_calI2 = np.arange(0,85,5) # bias current sweep range

        c2f_calI2 = []

        for n in range(len(bg_fine_calI2)):

            # set bias
            p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
            pyplane.Coach.BiasAddress.BAB_VB_N, \
            pyplane.Coach.BiasType.N, \
            pyplane.Coach.BiasGenMasterCurrent.I30nA, bg_fine_calI2[n])])

            time.sleep(0.5) # settle time

            # read c2f values for 0.1s duration
            c2f_calI2_temp = p.read_c2f_output(0.1)
            c2f_calI2.append(c2f_calI2_temp[5])   #set index for c2F to 5
        print(c2f_calI2)
```

```
np.savetxt('BUMP_c2f_calI1_vs_bg_fine_calI1.csv',[c2f_calI2,bg_fine_calI2], delimiter=
```

```
[2, 1071, 2078, 3096, 4009, 4983, 5908, 6708, 7534, 8442, 9602, 11874, 13771, 15673,
18456, 21593, 24415]
```

- Plot

In [ ]:
```
plt.rcParams.update({'font.size': 15})

c2f_calI2,bg_fine_calI2 = np.loadtxt('BUMP_c2f_calI1_vs_bg_fine_calI1.csv', delimiter=

Ib_calI2 = bg_fine_calI2/256*30

plt.plot(Ib_calI2,c2f_calI2,'b+')

plt.xlabel('$I_b$ [nA]')
plt.ylabel('C2F [Hz]')
plt.legend(['C2F'],prop={'size': 14})
plt.title('Fig. 11: C2F values vs. $I_b$ for $V_2 \gg V_1$.')
plt.grid()
plt.show()
```



- Save data

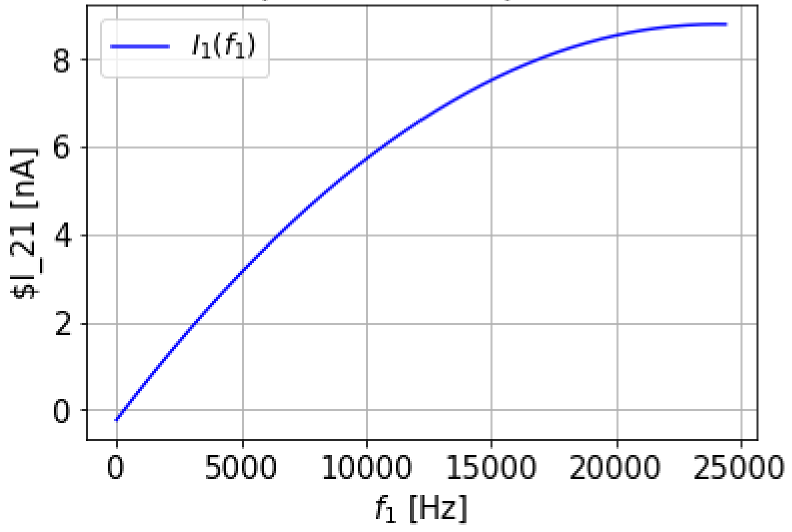
- Extract the function $I_1(f_1)$ (Hint: use higher order polynomial to increase accuracy)

In [ ]:
```
# fit quadratic polynomial to C2F vs Ib data
a2_I1cal,a1_I1cal,a0_I1cal = np.polyfit(c2f_calI2[:16],Ib_calI2[:16],2)

print(a0_I1cal)
print(a1_I1cal)
print(a2_I1cal)

range_I2cal = np.arange(1,c2f_calI2[16],14) # select interpolation interval, omitting
print(c2f_calI2[16])
print(range_I2cal)
```

```
plt.plot(range_I2cal,a0_I1cal+a1_I1cal*range_I2cal+a2_I1cal*range_I2cal**2,'b-')

plt.xlabel('$f_1$ [Hz]')
plt.ylabel('$I_21 [nA]')
plt.legend(['$I_1(f_1)$'],prop={'size': 14})
plt.title('Fig. 4: Quadratic interpolation of $I_1$ plotted as a function of $f_1$. ')
plt.grid()
plt.show()
```

```
-0.2369772208761792
0.0007511345289114774
-1.560959217343982e-08
24415.0
[1.0000e+00 1.5000e+01 2.9000e+01 ... 2.4375e+04 2.4389e+04 2.4403e+04]
```

Fig. 4: Quadratic interpolation of $I_1$ plotted as a function of $f_1$.



6.2.2 Calibration C2F response for I2

- Set fixed voltages for V_1 and V_2

In [ ]:
```
p.set_voltage(pyplane.DacChannel.AIN12,0.2) # V1 = 0.2
p.set_voltage(pyplane.DacChannel.AIN13,0.7) # V2 = 0.7
```

Out[ ]:
```
0.698533833026886
```

Set V_1 and V_2 such that $V_1 \ll V_2$.

- Data aquisition (Hint: linear range $I \leq 10$ nA)

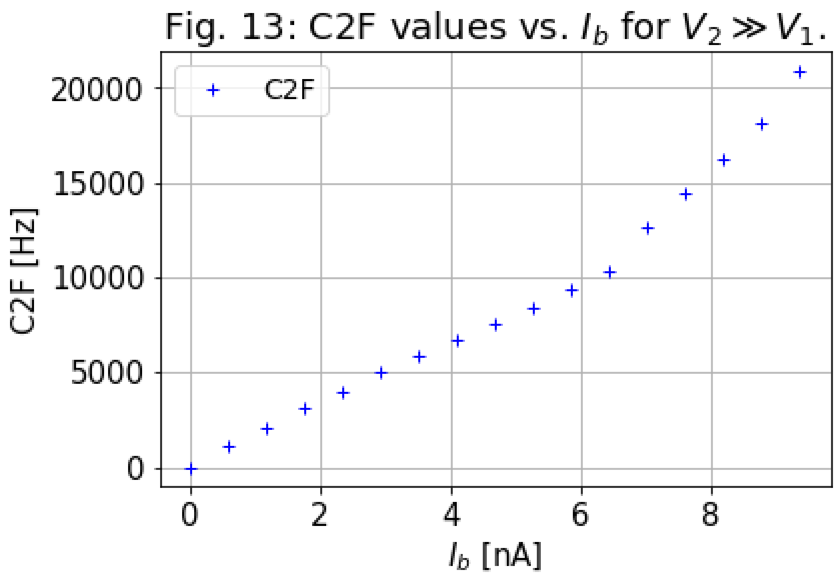
In [ ]:
```
import pyplane
import numpy as np
import time
import matplotlib.pyplot as plt
# your code

bg_fine_calI2 = np.arange(0,85,5) # bias current sweep range
```

```
c2f_calI2 = []

for n in range(len(bg_fine_calI2)):

    # set bias
    p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.BAB_VB_N, \
    pyplane.Coach.BiasType.N, \
    pyplane.Coach.BiasGenMasterCurrent.I30nA, bg_fine_calI2[n])])

    time.sleep(0.5) # settle time

    # read c2f values for 0.1s duration
    c2f_calI2_temp = p.read_c2f_output(0.1)
    c2f_calI2.append(c2f_calI2_temp[6])   #set index for c2F to 5
print(c2f_calI2)

np.savetxt('BUMP_c2f_calI2_vs_bg_fine_calI2.csv',[c2f_calI2,bg_fine_calI2], delimiter=
```

```
[2, 1069, 2077, 3089, 3998, 4984, 5903, 6704, 7515, 8453, 9346, 10362, 12605, 14428,
16284, 18152, 20907]
```

- Plot

In [ ]:
```
plt.rcParams.update({'font.size': 15})

c2f_calI2,bg_fine_calI2 = np.loadtxt('BUMP_c2f_calI2_vs_bg_fine_calI2.csv', delimiter=

Ib_calI2 = bg_fine_calI2/256*30

plt.plot(Ib_calI2,c2f_calI2,'b+')

plt.xlabel('$I_b$ [nA]')
plt.ylabel('C2F [Hz]')
plt.legend(['C2F'],prop={'size': 14})
plt.title('Fig. 13: C2F values vs. $I_b$ for $V_2 \gg V_1$.')
plt.grid()
plt.show()
```



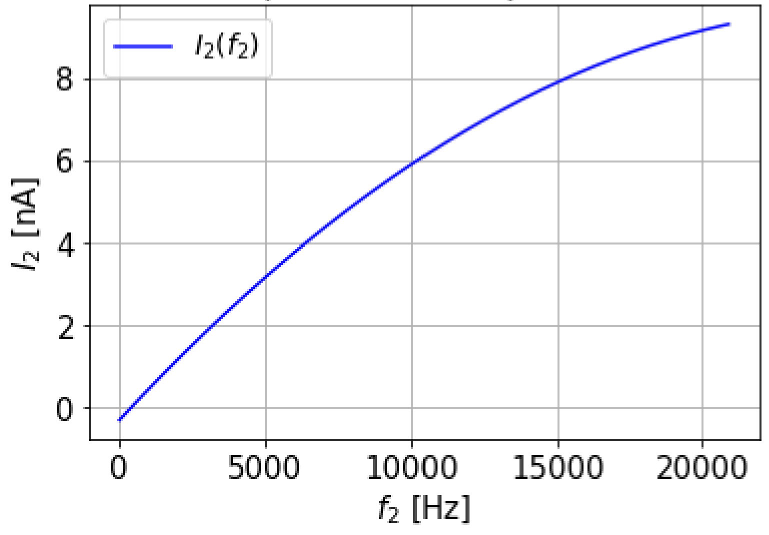
- Save data

- Extract the function $I_2(f_2)$ (Hint: use higher order polynomial to increase accuracy)

In [ ]:
```
# fit quadratic polynomial to C2F vs Ib data
a2_I2cal,a1_I2cal,a0_I2cal = np.polyfit(c2f_calI2[:16],Ib_calI2[:16],2)

print(a0_I2cal)
print(a1_I2cal)
print(a2_I2cal)

range_I2cal = np.arange(1,c2f_calI2[16],14) # select interpolation interval, omitting
print(c2f_calI2[16])
print(range_I2cal)
plt.plot(range_I2cal,a0_I2cal+a1_I2cal*range_I2cal+a2_I2cal*range_I2cal**2,'b-')

plt.xlabel('$f_2$ [Hz]')
plt.ylabel('$I_2$ [nA]')
plt.legend(['$I_2(f_2)$'],prop={'size': 14})
plt.title('Fig. 4: Quadratic interpolation of $I_2$ plotted as a function of $f_2$. ')
plt.grid()
plt.show()
```

```
-0.29490289497710975
0.0007669817457405208
-1.466188759782272e-08
20907.0
[1.0000e+00 1.5000e+01 2.9000e+01 ... 2.0875e+04 2.0889e+04 2.0903e+04]
```

Fig. 4: Quadratic interpolation of $I_2$ plotted as a function of $f_2$.



6.2.3 Calibration C2F response for Iout

- Set fixed voltages for V_1 and V_2

In [ ]:
```
p.set_voltage(pyplane.DacChannel.AIN12,0.5) # V1 = 0.5
p.set_voltage(pyplane.DacChannel.AIN13,0.5) # V2 = 0.5
```

Out[ ]:
```
0.49970680475234985
```

Set $V_1 \neq 0$ and $V_2 \neq 0$ such that $V_1 - V_2 = 0$.

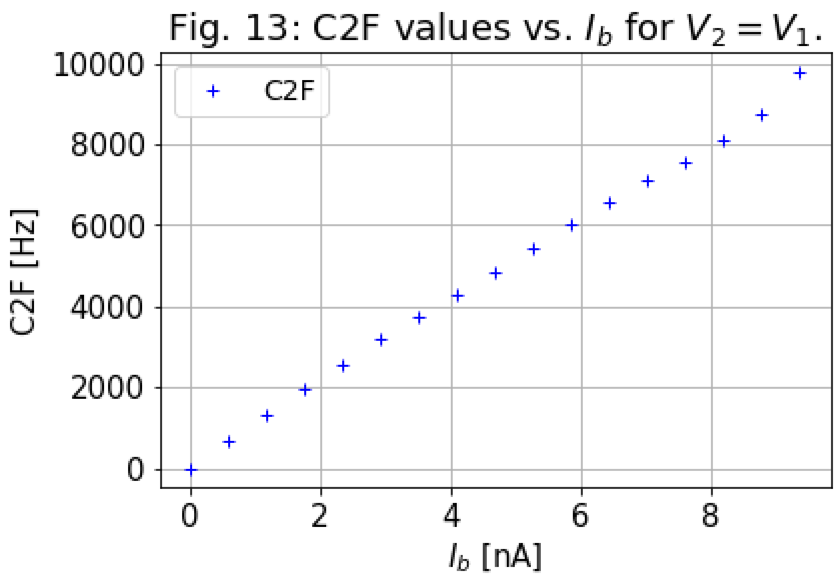
- Data aquisition (Hint: linear range $I \leq 10$ nA)

In [ ]:
```
import pyplane
import numpy as np
import time
import matplotlib.pyplot as plt
# your code

bg_fine_calI2 = np.arange(0,85,5) # bias current sweep range

c2f_calI2 = []

for n in range(len(bg_fine_calI2)):

    # set bias
    p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
    pyplane.Coach.BiasAddress.BAB_VB_N, \
    pyplane.Coach.BiasType.N, \
    pyplane.Coach.BiasGenMasterCurrent.I30nA, bg_fine_calI2[n])])

    time.sleep(0.5) # settle time

    # read c2f values for 0.1s duration
    c2f_calI2_temp = p.read_c2f_output(0.1)
    c2f_calI2.append(c2f_calI2_temp[7])   #set index for c2F to 5
print(c2f_calI2)

np.savetxt('BUMP_c2f_calIOUT_vs_bg_fine_calIOUT.csv',[c2f_calI2,bg_fine_calI2], delimi
```

```
[2, 683, 1323, 1950, 2572, 3204, 3736, 4279, 4840, 5426, 6002, 6571, 7105, 7573, 812
5, 8733, 9798]
```

- Plot

In [ ]:
```
plt.rcParams.update({'font.size': 15})

c2f_calI2,bg_fine_calI2 = np.loadtxt('BUMP_c2f_calIOUT_vs_bg_fine_calIOUT.csv', delimi

Ib_calI2 = bg_fine_calI2/256*30

plt.plot(Ib_calI2,c2f_calI2,'b+')

plt.xlabel('$I_b$ [nA]')
plt.ylabel('C2F [Hz]')
plt.legend(['C2F'],prop={'size': 14})
plt.title('Fig. 13: C2F values vs. $I_b$ for $V_2 = V_1$.')
plt.grid()
plt.show()
```



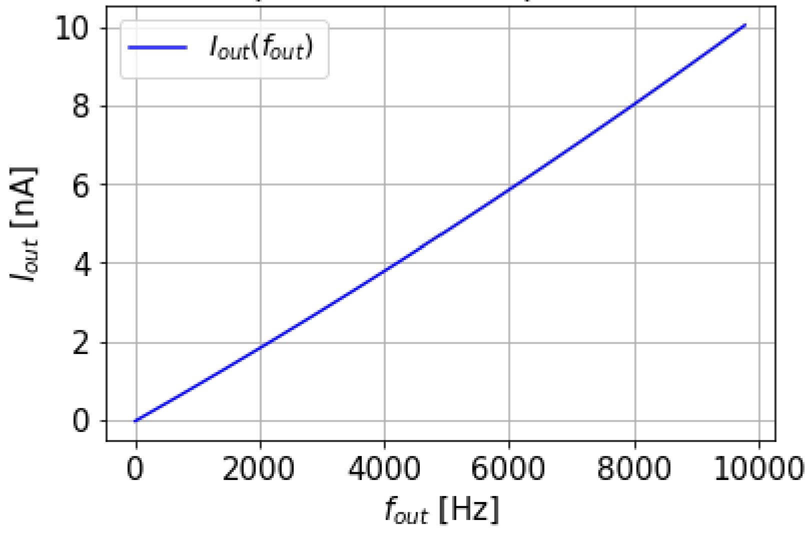
- Save data

- Extract the function $I_{out}\left(f_{out}\right)$ (Hint: use higher order polynomial to increase accuracy)

In [ ]:
```
# fit quadratic polynomial to C2F vs Iout data
a2_Ioutcal,a1_Ioutcal,a0_Ioutcal = np.polyfit(c2f_calI2[:16],Ib_calI2[:16],2)

print(a0_Ioutcal)
print(a1_Ioutcal)
print(a2_Ioutcal)

range_I2cal = np.arange(1,c2f_calI2[16],14) # select interpolation interval, omitting
print(c2f_calI2[16])
print(range_I2cal)
plt.plot(range_I2cal,a0_Ioutcal+a1_Ioutcal*range_I2cal+a2_Ioutcal*range_I2cal**2,'b-')

plt.xlabel('$f_{out}$ [Hz]')
plt.ylabel('$I_{out}$ [nA]')
plt.legend(['$I_{out}(f_{out})$'],prop={'size': 14})
plt.title('Fig. 4: Quadratic interpolation of $I_{out}$ plotted as a function of $f_{o
plt.grid()
plt.show()
```

```
-0.03925327296860514
0.0009044546933703591
1.3050587600350514e-08
9798.0
[1.000e+00 1.500e+01 2.900e+01 4.300e+01 5.700e+01 7.100e+01 8.500e+01
 9.900e+01 1.130e+02 1.270e+02 1.410e+02 1.550e+02 1.690e+02 1.830e+02
 1.970e+02 2.110e+02 2.250e+02 2.390e+02 2.530e+02 2.670e+02 2.810e+02
 2.950e+02 3.090e+02 3.230e+02 3.370e+02 3.510e+02 3.650e+02 3.790e+02
 3.930e+02 4.070e+02 4.210e+02 4.350e+02 4.490e+02 4.630e+02 4.770e+02
 4.910e+02 5.050e+02 5.190e+02 5.330e+02 5.470e+02 5.610e+02 5.750e+02
 5.890e+02 6.030e+02 6.170e+02 6.310e+02 6.450e+02 6.590e+02 6.730e+02
 6.870e+02 7.010e+02 7.150e+02 7.290e+02 7.430e+02 7.570e+02 7.710e+02
 7.850e+02 7.990e+02 8.130e+02 8.270e+02 8.410e+02 8.550e+02 8.690e+02
 8.830e+02 8.970e+02 9.110e+02 9.250e+02 9.390e+02 9.530e+02 9.670e+02
 9.810e+02 9.950e+02 1.009e+03 1.023e+03 1.037e+03 1.051e+03 1.065e+03
 1.079e+03 1.093e+03 1.107e+03 1.121e+03 1.135e+03 1.149e+03 1.163e+03
 1.177e+03 1.191e+03 1.205e+03 1.219e+03 1.233e+03 1.247e+03 1.261e+03
 1.275e+03 1.289e+03 1.303e+03 1.317e+03 1.331e+03 1.345e+03 1.359e+03
 1.373e+03 1.387e+03 1.401e+03 1.415e+03 1.429e+03 1.443e+03 1.457e+03
 1.471e+03 1.485e+03 1.499e+03 1.513e+03 1.527e+03 1.541e+03 1.555e+03
 1.569e+03 1.583e+03 1.597e+03 1.611e+03 1.625e+03 1.639e+03 1.653e+03
 1.667e+03 1.681e+03 1.695e+03 1.709e+03 1.723e+03 1.737e+03 1.751e+03
 1.765e+03 1.779e+03 1.793e+03 1.807e+03 1.821e+03 1.835e+03 1.849e+03
 1.863e+03 1.877e+03 1.891e+03 1.905e+03 1.919e+03 1.933e+03 1.947e+03
 1.961e+03 1.975e+03 1.989e+03 2.003e+03 2.017e+03 2.031e+03 2.045e+03
 2.059e+03 2.073e+03 2.087e+03 2.101e+03 2.115e+03 2.129e+03 2.143e+03
 2.157e+03 2.171e+03 2.185e+03 2.199e+03 2.213e+03 2.227e+03 2.241e+03
 2.255e+03 2.269e+03 2.283e+03 2.297e+03 2.311e+03 2.325e+03 2.339e+03
 2.353e+03 2.367e+03 2.381e+03 2.395e+03 2.409e+03 2.423e+03 2.437e+03
 2.451e+03 2.465e+03 2.479e+03 2.493e+03 2.507e+03 2.521e+03 2.535e+03
 2.549e+03 2.563e+03 2.577e+03 2.591e+03 2.605e+03 2.619e+03 2.633e+03
 2.647e+03 2.661e+03 2.675e+03 2.689e+03 2.703e+03 2.717e+03 2.731e+03
 2.745e+03 2.759e+03 2.773e+03 2.787e+03 2.801e+03 2.815e+03 2.829e+03
 2.843e+03 2.857e+03 2.871e+03 2.885e+03 2.899e+03 2.913e+03 2.927e+03
 2.941e+03 2.955e+03 2.969e+03 2.983e+03 2.997e+03 3.011e+03 3.025e+03
 3.039e+03 3.053e+03 3.067e+03 3.081e+03 3.095e+03 3.109e+03 3.123e+03
 3.137e+03 3.151e+03 3.165e+03 3.179e+03 3.193e+03 3.207e+03 3.221e+03
 3.235e+03 3.249e+03 3.263e+03 3.277e+03 3.291e+03 3.305e+03 3.319e+03
 3.333e+03 3.347e+03 3.361e+03 3.375e+03 3.389e+03 3.403e+03 3.417e+03
 3.431e+03 3.445e+03 3.459e+03 3.473e+03 3.487e+03 3.501e+03 3.515e+03
 3.529e+03 3.543e+03 3.557e+03 3.571e+03 3.585e+03 3.599e+03 3.613e+03
 3.627e+03 3.641e+03 3.655e+03 3.669e+03 3.683e+03 3.697e+03 3.711e+03
 3.725e+03 3.739e+03 3.753e+03 3.767e+03 3.781e+03 3.795e+03 3.809e+03
 3.823e+03 3.837e+03 3.851e+03 3.865e+03 3.879e+03 3.893e+03 3.907e+03
 3.921e+03 3.935e+03 3.949e+03 3.963e+03 3.977e+03 3.991e+03 4.005e+03
 4.019e+03 4.033e+03 4.047e+03 4.061e+03 4.075e+03 4.089e+03 4.103e+03
 4.117e+03 4.131e+03 4.145e+03 4.159e+03 4.173e+03 4.187e+03 4.201e+03
 4.215e+03 4.229e+03 4.243e+03 4.257e+03 4.271e+03 4.285e+03 4.299e+03
 4.313e+03 4.327e+03 4.341e+03 4.355e+03 4.369e+03 4.383e+03 4.397e+03
 4.411e+03 4.425e+03 4.439e+03 4.453e+03 4.467e+03 4.481e+03 4.495e+03
 4.509e+03 4.523e+03 4.537e+03 4.551e+03 4.565e+03 4.579e+03 4.593e+03
 4.607e+03 4.621e+03 4.635e+03 4.649e+03 4.663e+03 4.677e+03 4.691e+03
 4.705e+03 4.719e+03 4.733e+03 4.747e+03 4.761e+03 4.775e+03 4.789e+03
 4.803e+03 4.817e+03 4.831e+03 4.845e+03 4.859e+03 4.873e+03 4.887e+03
 4.901e+03 4.915e+03 4.929e+03 4.943e+03 4.957e+03 4.971e+03 4.985e+03
 4.999e+03 5.013e+03 5.027e+03 5.041e+03 5.055e+03 5.069e+03 5.083e+03
 5.097e+03 5.111e+03 5.125e+03 5.139e+03 5.153e+03 5.167e+03 5.181e+03
 5.195e+03 5.209e+03 5.223e+03 5.237e+03 5.251e+03 5.265e+03 5.279e+03
 5.293e+03 5.307e+03 5.321e+03 5.335e+03 5.349e+03 5.363e+03 5.377e+03
 5.391e+03 5.405e+03 5.419e+03 5.433e+03 5.447e+03 5.461e+03 5.475e+03
 5.489e+03 5.503e+03 5.517e+03 5.531e+03 5.545e+03 5.559e+03 5.573e+03
 5.587e+03 5.601e+03 5.615e+03 5.629e+03 5.643e+03 5.657e+03 5.671e+03
 5.685e+03 5.699e+03 5.713e+03 5.727e+03 5.741e+03 5.755e+03 5.769e+03
 5.783e+03 5.797e+03 5.811e+03 5.825e+03 5.839e+03 5.853e+03 5.867e+03
 5.881e+03 5.895e+03 5.909e+03 5.923e+03 5.937e+03 5.951e+03 5.965e+03
 5.979e+03 5.993e+03 6.007e+03 6.021e+03 6.035e+03 6.049e+03 6.063e+03
 6.077e+03 6.091e+03 6.105e+03 6.119e+03 6.133e+03 6.147e+03 6.161e+03
 6.175e+03 6.189e+03 6.203e+03 6.217e+03 6.231e+03 6.245e+03 6.259e+03
 6.273e+03 6.287e+03 6.301e+03 6.315e+03 6.329e+03 6.343e+03 6.357e+03
 6.371e+03 6.385e+03 6.399e+03 6.413e+03 6.427e+03 6.441e+03 6.455e+03
 6.469e+03 6.483e+03 6.497e+03 6.511e+03 6.525e+03 6.539e+03 6.553e+03
 6.567e+03 6.581e+03 6.595e+03 6.609e+03 6.623e+03 6.637e+03 6.651e+03
 6.665e+03 6.679e+03 6.693e+03 6.707e+03 6.721e+03 6.735e+03 6.749e+03
 6.763e+03 6.777e+03 6.791e+03 6.805e+03 6.819e+03 6.833e+03 6.847e+03
 6.861e+03 6.875e+03 6.889e+03 6.903e+03 6.917e+03 6.931e+03 6.945e+03
 6.959e+03 6.973e+03 6.987e+03 7.001e+03 7.015e+03 7.029e+03 7.043e+03
 7.057e+03 7.071e+03 7.085e+03 7.099e+03 7.113e+03 7.127e+03 7.141e+03
 7.155e+03 7.169e+03 7.183e+03 7.197e+03 7.211e+03 7.225e+03 7.239e+03
 7.253e+03 7.267e+03 7.281e+03 7.295e+03 7.309e+03 7.323e+03 7.337e+03
 7.351e+03 7.365e+03 7.379e+03 7.393e+03 7.407e+03 7.421e+03 7.435e+03
 7.449e+03 7.463e+03 7.477e+03 7.491e+03 7.505e+03 7.519e+03 7.533e+03
 7.547e+03 7.561e+03 7.575e+03 7.589e+03 7.603e+03 7.617e+03 7.631e+03
 7.645e+03 7.659e+03 7.673e+03 7.687e+03 7.701e+03 7.715e+03 7.729e+03
 7.743e+03 7.757e+03 7.771e+03 7.785e+03 7.799e+03 7.813e+03 7.827e+03
 7.841e+03 7.855e+03 7.869e+03 7.883e+03 7.897e+03 7.911e+03 7.925e+03
 7.939e+03 7.953e+03 7.967e+03 7.981e+03 7.995e+03 8.009e+03 8.023e+03
 8.037e+03 8.051e+03 8.065e+03 8.079e+03 8.093e+03 8.107e+03 8.121e+03
 8.135e+03 8.149e+03 8.163e+03 8.177e+03 8.191e+03 8.205e+03 8.219e+03
 8.233e+03 8.247e+03 8.261e+03 8.275e+03 8.289e+03 8.303e+03 8.317e+03
 8.331e+03 8.345e+03 8.359e+03 8.373e+03 8.387e+03 8.401e+03 8.415e+03
 8.429e+03 8.443e+03 8.457e+03 8.471e+03 8.485e+03 8.499e+03 8.513e+03
 8.527e+03 8.541e+03 8.555e+03 8.569e+03 8.583e+03 8.597e+03 8.611e+03
 8.625e+03 8.639e+03 8.653e+03 8.667e+03 8.681e+03 8.695e+03 8.709e+03
 8.723e+03 8.737e+03 8.751e+03 8.765e+03 8.779e+03 8.793e+03 8.807e+03
 8.821e+03 8.835e+03 8.849e+03 8.863e+03 8.877e+03 8.891e+03 8.905e+03
 8.919e+03 8.933e+03 8.947e+03 8.961e+03 8.975e+03 8.989e+03 9.003e+03
 9.017e+03 9.031e+03 9.045e+03 9.059e+03 9.073e+03 9.087e+03 9.101e+03
 9.115e+03 9.129e+03 9.143e+03 9.157e+03 9.171e+03 9.185e+03 9.199e+03
 9.213e+03 9.227e+03 9.241e+03 9.255e+03 9.269e+03 9.283e+03 9.297e+03
 9.311e+03 9.325e+03 9.339e+03 9.353e+03 9.367e+03 9.381e+03 9.395e+03
 9.409e+03 9.423e+03 9.437e+03 9.451e+03 9.465e+03 9.479e+03 9.493e+03
 9.507e+03 9.521e+03 9.535e+03 9.549e+03 9.563e+03 9.577e+03 9.591e+03
 9.605e+03 9.619e+03 9.633e+03 9.647e+03 9.661e+03 9.675e+03 9.689e+03
 9.703e+03 9.717e+03 9.731e+03 9.745e+03 9.759e+03 9.773e+03 9.787e+03]
```

Fig. 4: Quadratic interpolation of I_{out} plotted as a function of f_{out}.



6.3 Basic measurement

- Assign common-mode voltage V_{cm}

```
In [ ]: Vcm_bm_bab = 0.9
```

- Set bias current

```
In [ ]: p.send_coach_events([pyplane.Coach.generate_biasgen_event(\
            pyplane.Coach.BiasAddress.BAB_VB_N, \
            pyplane.Coach.BiasType.N, \
            pyplane.Coach.BiasGenMasterCurrent.I30nA, 12)])
```

Set bias current to $I_b = w\frac{BG_{\text{fine}}}{256} I_{BG_{\text{master}}} = 3 \cdot \frac{12}{256} \cdot 30\text{nA} \approx 4.219\text{nA}$.

- Data aquisition
- You can follow the example below

```
In [ ]: import numpy as np
        import time

        # your code

        V1_Vcm_bm_bab  = np.arange(0.65,1.15,0.005) # V1 sweep range

        V2_Vcm_bm_bab  = []
        V1_Vcm_bm_set_bab  = []
        V2_Vcm_bm_set_bab  = []
        c2f_Vcm_I1_bm_bab  = []
        c2f_Vcm_I2_bm_bab  = []
        c2f_Vcm_Iout_bm_bab = []
```

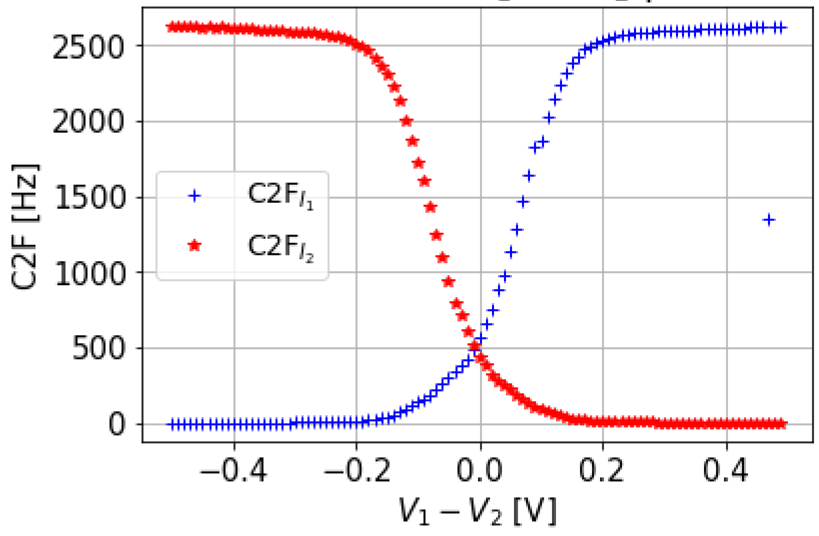
```
for n in range(len(V1_Vcm_bm_bab)):

    V2_Vcm_bm_bab.append(2*Vcm_bm_bab -V1_Vcm_bm_bab[n])

    p.set_voltage(pyplane.DacChannel.AIN12,V1_Vcm_bm_bab[n]) # V1
    p.set_voltage(pyplane.DacChannel.AIN13,V2_Vcm_bm_bab[n]) # V2

    time.sleep(0.5) # settle time

    V1_Vcm_bm_set_bab.append(p.get_set_voltage(pyplane.DacChannel.AIN12))
    V2_Vcm_bm_set_bab.append(p.get_set_voltage(pyplane.DacChannel.AIN13))

    # read c2f values
    c2f_Vcm_temp = p.read_c2f_output(0.1)
    c2f_Vcm_I1_bm_bab.append(c2f_Vcm_temp[5])
    c2f_Vcm_I2_bm_bab.append(c2f_Vcm_temp[6])
    c2f_Vcm_Iout_bm_bab.append(c2f_Vcm_temp[7])

print(V1_Vcm_bm_bab)
print(V2_Vcm_bm_bab)
print(c2f_Vcm_I1_bm_bab)
print(c2f_Vcm_I2_bm_bab)

data_Vcm_bm = [V1_Vcm_bm_bab,V2_Vcm_bm_bab,V1_Vcm_bm_set_bab,V2_Vcm_bm_set_bab,c2f_Vcm
np.savetxt('BAB_c2f_Vcm_bm_vs_V1_V2.csv', data_Vcm_bm, delimiter=',')
```

```
[0.65  0.655 0.66  0.665 0.67  0.675 0.68  0.685 0.69  0.695 0.7   0.705
 0.71  0.715 0.72  0.725 0.73  0.735 0.74  0.745 0.75  0.755 0.76  0.765
 0.77  0.775 0.78  0.785 0.79  0.795 0.8   0.805 0.81  0.815 0.82  0.825
 0.83  0.835 0.84  0.845 0.85  0.855 0.86  0.865 0.87  0.875 0.88  0.885
 0.89  0.895 0.9   0.905 0.91  0.915 0.92  0.925 0.93  0.935 0.94  0.945
 0.95  0.955 0.96  0.965 0.97  0.975 0.98  0.985 0.99  0.995 1.    1.005
 1.01  1.015 1.02  1.025 1.03  1.035 1.04  1.045 1.05  1.055 1.06  1.065
 1.07  1.075 1.08  1.085 1.09  1.095 1.1   1.105 1.11  1.115 1.12  1.125
 1.13  1.135 1.14  1.145]
[1.15, 1.145, 1.1400000000000001, 1.135, 1.13, 1.125, 1.12, 1.115, 1.109999999999999
9, 1.105, 1.1, 1.095, 1.0899999999999999, 1.085, 1.08, 1.075, 1.0699999999999998, 1.0
65, 1.06, 1.055, 1.0499999999999998, 1.045, 1.04, 1.035, 1.0299999999999998, 1.025,
1.02, 1.015, 1.0099999999999998, 1.005, 0.9999999999999999, 0.9949999999999999, 0.989
9999999999999, 0.9849999999999999, 0.9799999999999999, 0.9749999999999999, 0.96999999
99999999, 0.9649999999999999, 0.9599999999999999, 0.9549999999999998, 0.9499999999999
998, 0.9449999999999998, 0.9399999999999998, 0.9349999999999998, 0.9299999999999998,
0.9249999999999998, 0.9199999999999998, 0.9149999999999998, 0.9099999999999998, 0.904
9999999999998, 0.8999999999999998, 0.8949999999999998, 0.8899999999999998, 0.88499999
99999998, 0.8799999999999998, 0.8749999999999998, 0.8699999999999998, 0.8649999999999
998, 0.8599999999999998, 0.8549999999999998, 0.8499999999999998, 0.8449999999999998,
0.8399999999999997, 0.8349999999999997, 0.8299999999999997, 0.8249999999999997, 0.819
9999999999997, 0.8149999999999997, 0.8099999999999997, 0.8049999999999997, 0.79999999
99999996, 0.7949999999999997, 0.7899999999999998, 0.7849999999999997, 0.7799999999999
996, 0.7749999999999997, 0.7699999999999998, 0.7649999999999997, 0.7599999999999996,
0.7549999999999997, 0.7499999999999998, 0.7449999999999997, 0.7399999999999995, 0.734
9999999999997, 0.7299999999999998, 0.7249999999999996, 0.7199999999999995, 0.71499999
99999996, 0.7099999999999997, 0.7049999999999996, 0.6999999999999995, 0.6949999999999
996, 0.6899999999999997, 0.6849999999999996, 0.6799999999999995, 0.6749999999999996,
0.6699999999999997, 0.6649999999999996, 0.6599999999999995, 0.6549999999999996]
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 5, 6,
7, 9, 12, 15, 20, 26, 34, 41, 53, 69, 88, 112, 136, 157, 188, 221, 259, 300, 341, 37
5, 420, 489, 564, 654, 752, 892, 975, 1136, 1288, 1470, 1644, 1830, 1871, 2028, 2147,
2245, 2326, 2389, 2423, 2477, 2492, 2515, 2533, 2548, 2562, 2567, 2578, 2582, 2588, 2
591, 2592, 2596, 2594, 2596, 2602, 2605, 2602, 2605, 2610, 2610, 2613, 2611, 2612, 26
16, 2618, 2617, 2620, 2621, 2625, 1347, 2625, 2628]
[2627, 2628, 2621, 2627, 2623, 2618, 2623, 2618, 2621, 2614, 2613, 2609, 2606, 2607,
2605, 2599, 2601, 2599, 2599, 2592, 2589, 2587, 2583, 2581, 2578, 2574, 2564, 2553, 2
542, 2528, 2504, 2492, 2461, 2412, 2360, 2311, 2234, 2131, 2005, 1870, 1726, 1603, 14
36, 1247, 1092, 936, 792, 714, 608, 513, 438, 375, 321, 278, 251, 215, 183, 154, 127,
104, 90, 71, 56, 43, 33, 26, 21, 16, 13, 10, 8, 6, 5, 5, 4, 3, 3, 3, 3, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

- Plot raw data (frequency)

In [ ]:
```python
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})

V1_Vcm_bm,V2_Vcm_bm,V1_Vcm_bm_set,V2_Vcm_bm_set,c2f_Vcm_I1_bm,c2f_Vcm_I2_bm = np.loadt

range_V1V2_bm = V1_Vcm_bm - V2_Vcm_bm

plt.plot(range_V1V2_bm,c2f_Vcm_I1_bm,'b+',range_V1V2_bm,c2f_Vcm_I2_bm,'r*')

plt.xlabel('$V_1-V_2$ [V]')
plt.ylabel('C2F [Hz]')
plt.legend(['C2F$_{I_1}$','C2F$_{I_2}$'],prop={'size': 14})
plt.title('Fig. 9: Measured C2F data for $I_1$ and $I_2$ plotted over $V_1-V_2$.')
plt.grid()
plt.show()
```

Fig. 9: Measured C2F data for $I_1$ and $I_2$ plotted over $V_1 - V_2$.



- Save raw data

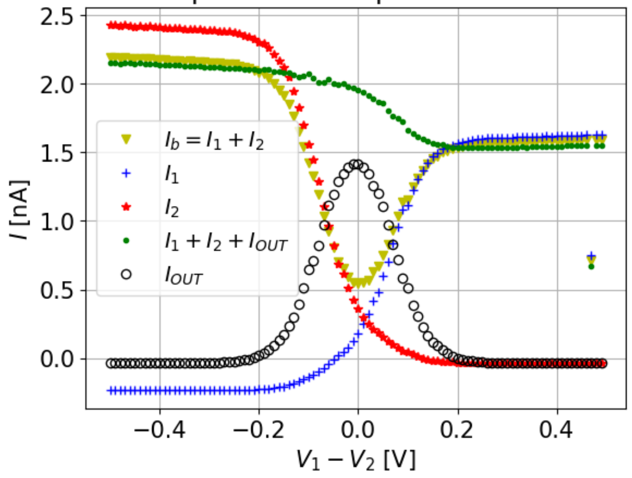
- Convert frequency to current

In [ ]:
```
# Use bias measurements
I1_bm_bab = a0_I1cal+a1_I1cal*np.array(c2f_Vcm_I1_bm_bab)+a2_I1cal*np.array(c2f_Vcm_I1
I2_bm_bab = a0_I2cal+a1_I2cal*np.array(c2f_Vcm_I2_bm_bab)+a2_I2cal*np.array(c2f_Vcm_I2
I_OUT_bm_bab = a0_Ioutcal+a1_I2cal*np.array(c2f_Vcm_Iout_bm_bab)+a2_Ioutcal*np.array(c
```

- Plot $I_1$, $I_2$, I_{out}, $I_1 + I_2$, $I_1 + I_2 + I_{out}$

In [ ]:
```
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})

range_V1V2_bm = V1_Vcm_bm_bab - V2_Vcm_bm_bab

plt.plot(range_V1V2_bm,I1_bm_bab+I2_bm_bab,'yv',label='$I_b=I_1+I_2$')
plt.plot(range_V1V2_bm,I1_bm_bab,'b+',label='$I_1$')
plt.plot(range_V1V2_bm,I2_bm_bab,'r*',label='$I_2$')
plt.plot(range_V1V2_bm,I1_bm_bab+I2_bm_bab+I_OUT_bm_bab,'g.',label='$I_1+I_2+I_{OUT}$'
plt.plot(range_V1V2_bm,I_OUT_bm_bab,'ko',label='$I_{OUT}$',mfc = "none")

plt.xlabel('$V_1-V_2$ [V]')
plt.ylabel('$I$ [nA]')
plt.legend(prop={'size': 14})
plt.title('Fig. 17: Interpolated differential pair currents plotted over the voltage d
plt.grid()
plt.show()
```

Fig. 17: Interpolated differential pair currents plotted over the voltage difference $V_1 - V_2$.



6.4 Comparison with calculation (optional)

- Based on prelab question 4c and the transistor W/L ratios shown in the schematic, does the measured ratio of maximum bump current to bias current accord with your measurement? Comment on possible reasons for any discrepancy between the fit and what you expect from the known transistor geometry. These effects are known to the logic guys as the short- and narrow-channel threshold shift effects.
- You can follow the example below (but you need to change all the variables names)

In [ ]:
```
import numpy as np
V1_Vcm_bm_bab,V2_Vcm_bm_bab,V1_Vcm_bm_set_bab,V2_Vcm_bm_set_bab,c2f_Vcm_I1_bm_bab,c2f_

c2f_calIout_bab,bg_fine_calIout_bab = np.loadtxt('c2f_calI1_vs_bg_fine_calIout_bab.csv
c2f_calI2_bab,bg_fine_calI2_bab = np.loadtxt('c2f_calI1_vs_bg_fine_calI2_bab.csv', del
c2f_calI1_bab,bg_fine_calI1_bab = np.loadtxt('c2f_calI1_vs_bg_fine_calI1_bab.csv', del

Ib_calIout_bab = bg_fine_calIout_bab/256*30*3
Ib_calI2_bab = bg_fine_calI2_bab/256*30*3
Ib_calI1_bab = bg_fine_calI1_bab/256*30*3

a2_Ioutcal_bab,a1_Ioutcal_bab,a0_Ioutcal_bab = np.polyfit(c2f_calIout_bab[:64],4/5*Ib_
a2_I2cal_bab,a1_I2cal_bab,a0_I2cal_bab = np.polyfit(c2f_calI2_bab[:64],Ib_calI2_bab[:6
a2_I1cal_bab,a1_I1cal_bab,a0_I1cal_bab = np.polyfit(c2f_calI1_bab[:64],Ib_calI1_bab[:6

I1_bm_bab = a0_I1cal_bab+a1_I1cal_bab*np.array(c2f_Vcm_I1_bm_bab)+a2_I1cal_bab*np.arra
I2_bm_bab = a0_I2cal_bab+a1_I2cal_bab*np.array(c2f_Vcm_I2_bm_bab)+a2_I2cal_bab*np.arra
Iout_bm_bab = a0_Ioutcal_bab+a1_Ioutcal_bab*np.array(c2f_Vcm_Iout_bm_bab)+a2_Ioutcal_b

Ib_bm_bab = I1_bm_bab + I2_bm_bab + Iout_bm_bab

print('Index of currents at V_2-V_1 = 0: ',int(len(Ib_bm_bab)/2-1))
print('Ratio of measured I_out,max to I_b: ',Iout_bm_bab[49]/Ib_bm_bab[49])
print('Ratio of measured I_out,max to I_b: ',Iout_bm_bab[49]/Ib_bm_bab[69])
```

In the prelab, it was determined that the bump current assumes its maximum value $I_{out,max}$ when $V_1 = V_2$. The corresponding ratio to the bias current I_b is

$$\frac{I_{out,max}}{I_b} = \frac{r}{r+4},$$

where $r = r_1 = r_2$ is the W/L-ratio of the respective input-output transistor pairs.

Using the given transistor geometries, it can thus be determined that the theoretical ratio of the maximum bump current to the bias current is

$$r = \frac{\frac{16u}{1u}}{\frac{1u}{1u}} = 16$$

$$\Rightarrow \left(\frac{I_{out,max}}{I_b}\right)_{\text{theo.}} = \frac{4}{5} = 0.8.$$

In contrast, the measured ratio of the maximum bump current to the bias current is

$$\left(\frac{I_{out,max}}{I_b}\right)_{\text{meas.}} \approx 0.7759.$$

The deviation between the theoretical and measured ratios is thus

$$1 - \frac{\left(\frac{I_{out,max}}{I_b}\right)_{\text{meas.}}}{\left(\frac{I_{out,max}}{I_b}\right)_{\text{theo.}}} \approx 0.0301 = 3.01\%.$$

The theoretical value therefore approximates the measured values quiet well. The remaining error can be explained by the short- and narrow-channel treshold shift effects:

- Narrow-Channel Effect:

This effect occurs when the width W of a transistor is small. Due to the extension of the depletion region underneath the gate toward the sides, some field lines from the gate end in depletion region under the oxide instead of the depletion region underneath the gate. This increases the perceived treshold voltage.

- Short-Channel Effects:

These effect occur when the width L of a transistor is small, and are mainly the result of the decrease effective channel length with rising channel current (Early effect). This increases the voltage drop across the pinchoff-region, which increases the electric field around the drain. Once velocity saturation of the carriers occurs, this causes the transistor current to decrease and hot-carrier effects to occur with increasing field around the drain. This once again results in a perceived increase of the treshold voltage of the transistor.

The increased treshold voltage caused by these two effects implies that I_b has to be slightly larger to bias the transistors in the circuit, explaining why it is larger relative to the theoretical predictions.

Note that these effect becomes particularly noticeable near $|V_1 - V_2| = 0\mathrm{V}$, where they occur for all transistors simultaneously (compared to $|V_1 - V_2| \gg 0\mathrm{V}$, where noteworthy current is only flowing through a subset of all transistors in the circuit). (?)

- Hand in the plotted subthreshold curves along with the fit to the antibump current.
- You can follow the example below (but you need to change all the variables names)

In [ ]:
```
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 15})

range_V1V2_bm_bab  = V2_Vcm_bm_bab  - V1_Vcm_bm_bab
#eval_points = np.arange(-0.5,0.5,0.01)

a1_I1V1V2_bab,a0_I1V1V2_bab = np.polyfit(range_V1V2_bm_bab[50:64],I1_bm_bab[50:64],1)
a1_I2V1V2_bab,a0_I2V1V2_bab = np.polyfit(range_V1V2_bm_bab[37:50],I2_bm_bab[37:50],1)
a2_IabV1V2_bab, a1_IabV1V2_bab,a0_IabV1V2_bab = np.polyfit(range_V1V2_bm_bab[37:64],I1

I1_interp_bab = a0_I1V1V2_bab + a1_I1V1V2_bab*np.array(range_V1V2_bm_bab[50:64])
I2_interp_bab = a0_I2V1V2_bab + a1_I2V1V2_bab*np.array(range_V1V2_bm_bab[37:50])
Iab_interp_bab = a0_IabV1V2_bab + a1_IabV1V2_bab*np.array(range_V1V2_bm_bab[37:64]) +


plt.plot(range_V1V2_bm_bab[30:71] ,I1_bm_bab[30:71] ,'b+',alpha=0.2)
plt.plot(range_V1V2_bm_bab[30:71] ,I2_bm_bab[30:71] ,'r*',alpha=0.2)
plt.plot(range_V1V2_bm_bab[30:71] ,I1_bm_bab[30:71] +I2_bm_bab[30:71] ,'y.')
plt.plot(range_V1V2_bm_bab[50:64] ,I1_interp_bab,'b-',alpha=0.6)
plt.plot(range_V1V2_bm_bab[37:50] ,I2_interp_bab,'r-',alpha=0.6)
plt.plot(range_V1V2_bm_bab[37:64] ,Iab_interp_bab,'g-')

#plt.plot(eval_points ,a0_I2cal_bab+a1_I2cal_bab*np.array(eval_points)+a2_I2cal_bab*np
#plt.plot(eval_points ,a0_I1cal_bab+a1_I1cal_bab*np.array(eval_points)+a2_I1cal_bab*np

plt.xlabel('$V_2-V_1$ [V]')
plt.ylabel('$I$ [nA]')
plt.legend(['$I_1$','$I_2$','$I_1+I_2$','$I_1$ Linear Fit','$I_2$ Linear Fit','$I_1+I_
plt.title('Fig. 19: Quadratically fitted subtreshold antibump current plotted over the
plt.grid()
plt.show()
```