MEMSの振幅制御速度向上

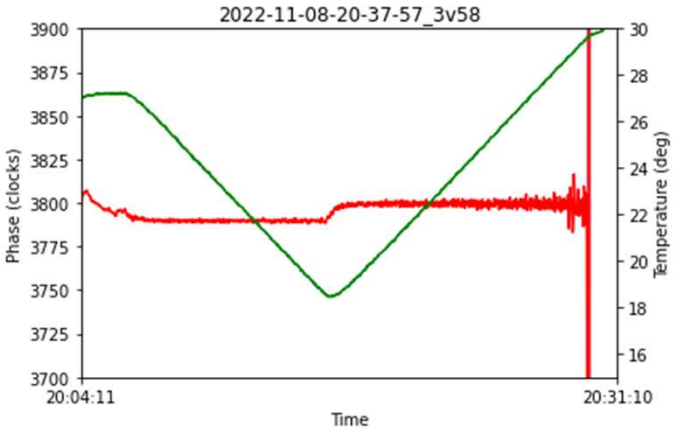
Python高速化の予備検討

2022/11/16

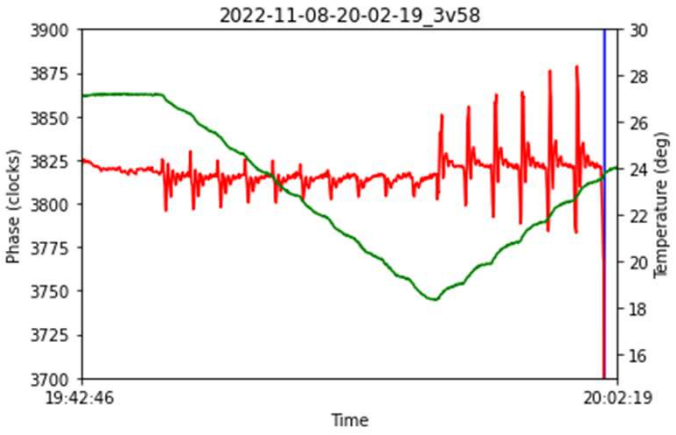
園田慎一郎

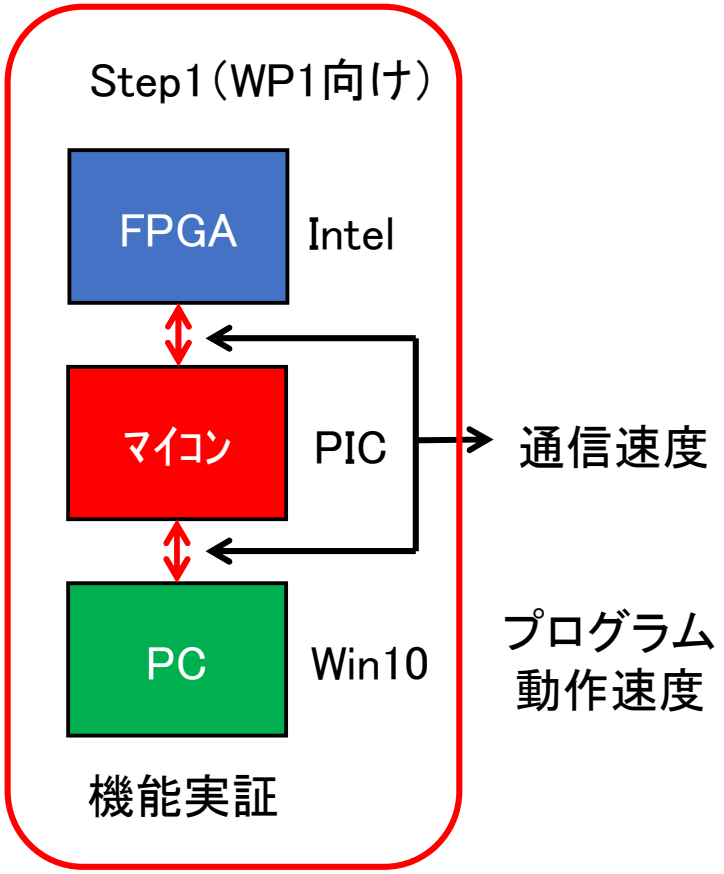
制御速度の問題

ΔT=0.1℃/6sec



ΔT=1℃/60sec





Ver.2
70W 500x700mm

真空PKGにおいてMEMSの特性が温度に敏感になる。
温度変化への追従するための制御速度（サンプリング周波数）を早くする必要がある。

Ver3ボードの開発方法

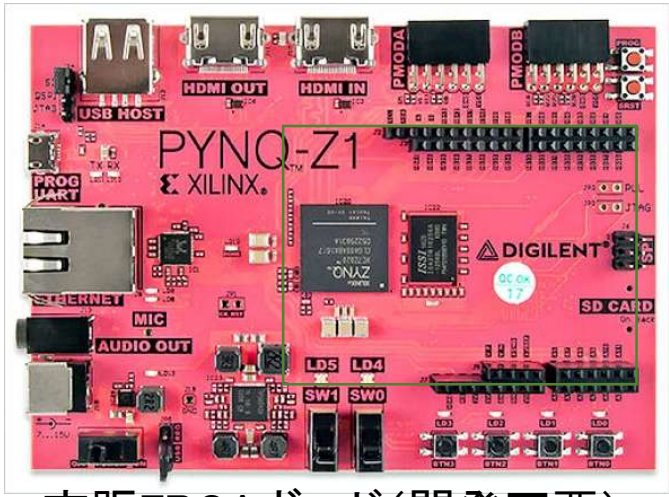
STEP3

FPGA

MEMS駆動用

FPGA内に
ﾏｲｺﾝを作成
し1ﾁｯﾌﾟ化

Ver.3
5W
53x69mm



市販FPGAボード（開発不要）
87x140 mm

＋

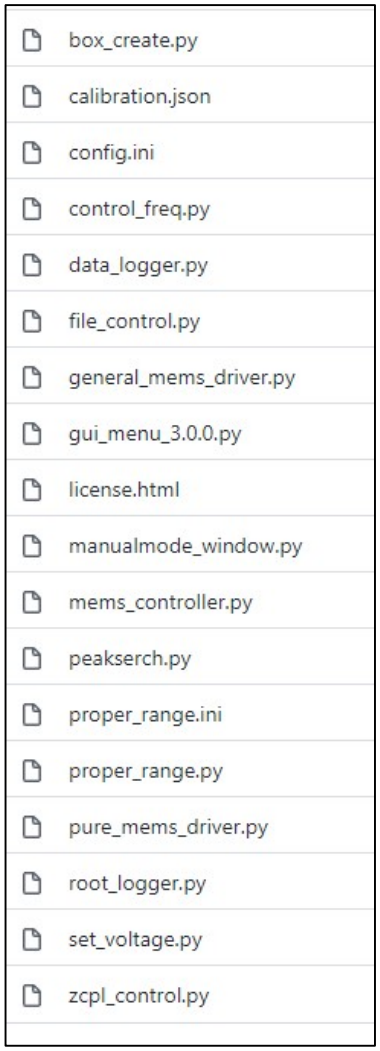
周辺回路

特注周辺回路ボード
53x69 mm

市販FPGAボードを活用し特注周辺回路ボードを直ぐに開発できるようにして開発納期を短縮する。
XILINX/PYNKを用いてFPGA内CPUをLinuxブートしてVer2ボードで開発したPythonコードを活用できるようにする。
またHDMI入力を活用し将来画像処理やレーザ描画の機能搭載も可能とする。
M社向けPOE0ボードはXILINX/PYNKの部品を使用し、機能・サイズを削減した特注FPGAボードを開発する。
（市販FPGAボードと特注FPGAボードのソフトウエアは完全に共通化される。）

マイコンとFPGAの通信バスが太くなり通信速度の問題は解消する。
一方ソフトウエアは、Pythonコードの活用が前提なので
プログラム速度は改善しない。

MEMS駆動プログラム例



14個のPythonモジュール
から構成されている。

```
848     Status.Ch2_Amp = Conditions.Ch2_Lim_Amp.get()
849
850     Status.message = "Now, operation."
851
852     return Status
853
854 def mems_zc_calibration(mems_zc, Set, Target, Conditions):
855     logger.info("mems zc calibration  {}".format(Target.Sample_Name.get()))
856     # time.sleep(0.1)
857     mems_zc.zcpl_execution()
858     # time.sleep(0.1)
859
860     ret = 0
861
862     if ret != 0:
863         messagebox.showerror("Error message", "Failed to set Zero Cross Pulse.\nError Code is " + str(ret))
864
865
866 def mems_zc_manual(mems_zc, Target, Conditions):
867     logger.info("mems zc manual  {}".format(Target.Sample_Name.get()))
868     check_bit = mems_zc.set_manual_delay_clk(int(Conditions.Ch2_Phase.get()), int(Conditions.Ch1_Phase.get()))
869
870     return check_bit
871
872
873 def freq2word(freq):
874         return freq / (160_000_000) * 2 ** 64
875
876 def word2freq(word):
877         return word / (2**64) * 160_000_000
878
879 if __name__ == '__main__':
880
881     # Tk instance create
882     root  = tk.Tk()
883
884     # window size
885     #root.geometry("680x540") # Linux
886     root.geometry("580x460") # Windows
887     root.resizable(0,0)
888     root.protocol('WM_DELETE_WINDOW', (lambda: 'pass')())
889
890     # title
891     root.title("MEMS driving system for Phase_3 ver.2.2.0")
892
893     # frame create
894     app = Gui_menu(master=root)
895
896     # window mainloop
897     app.mainloop()
```

1個のPythonモジュールは数百行程度

Pythonプログラムの高速化

等差数列 $\sum_{i=1}^{n} i$ n = 100000000 を計算した。

4.915 seconds

```python
# python
def sum_py(n):
    i = 0

    for j in range(n + 1):
        i  = i + j

    return i
```

0.031 seconds

```c
// C
#include <stdio.h>
long long sum_c(long long n)
{
    long long i = 0;
    for (long long j = 1; j <= n; j ++)
    {
        i = i + j;
    }
    return i;
}
```

Python
n = 100000000
Sigma(1 to n) = 5000000050000000
8 function calls in 4.915 seconds

Ordered by: internal time

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	4.915	4.915	4.915	4.915	test_py.py:2(sum)
2	0.000	0.000	0.000	0.000	{built-in method builtins.print}
1	0.000	0.000	4.915	4.915	{built-in method builtins.exec}
1	0.000	0.000	4.915	4.915	<string>:1(<module>)
1	0.000	0.000	4.915	4.915	main.py:21(main_py)
1	0.000	0.000	0.000	0.000	{method 'format' of 'str' objects}
1	0.000	0.000	0.000	0.000	{method 'disable' of '_lsprof.Profiler' objects}

Ctypes
n = 100000000
Sigma(1 to n) = 5000000050000000
7 function calls in 0.031 seconds

Ordered by: internal time

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.031	0.031	0.031	0.031	main.py:29(main_dll)
2	0.000	0.000	0.000	0.000	{built-in method builtins.print}
1	0.000	0.000	0.031	0.031	{built-in method builtins.exec}
1	0.000	0.000	0.000	0.000	{method 'format' of 'str' objects}
1	0.000	0.000	0.031	0.031	<string>:1(<module>)
1	0.000	0.000	0.000	0.000	{method 'disable' of '_lsprof.Profiler' objects}

Pythonを高速化はC言語への移植で実現できる。
しかし移植の負荷が大きく原理検証実験と並行して進めることは現実的でない。

Cythonを用いた高速化の検討

Cython（サイソン）は、C言語によるPythonの拡張モジュールの作成の労力を軽減することを目的として開発されたプログラミング言語である。その言語仕様はほとんどPythonと同じだが、Cの関数を直接呼び出したり、C言語の変数の型やクラスを宣言できるなどの拡張が行われている。Cythonの処理系ではソースファイルをCのコードに変換し、コンパイルすればPythonの拡張モジュールになる。必要部分（処理の遅いところ）のみをCythonに書き換えてPythonを改変していくことができる。

4.915 seconds

```python
# python
def sum_py(n):
    i = 0

    for j in range(n + 1):
        i  = i + j

    return i
```

0.031 seconds

```cython
#cython
def sum_cy(long long n):
    cdef long long i = 0, j

    for j in range(n + 1):
        i  = i + j

    return i
```

0.031 seconds

```c
// C
#include <stdio.h>
long long sum_c(long long n)
{
    long long i = 0;
    for (long long j = 1; j <= n; j ++)
    {
        i = i + j;
    }
    return i;
}
```

Cythonの処理速度は、C言語にほぼ匹敵することを確認した。

まとめ

・原理検証段階でのプログラムの高速化はCythonの活用が最適と考える。
・原理検証終了後にC言語への移植を実施するのが良いと考える。

今後

・Cythonの習得（クラス継承等の具体的な記法の取得）
・現状のプログラムの処理時間とボトルネックの解析
・プログラム速度がボトルネックの部分のCython化を検討する。