2022 年度 3 回生前期学生実験 HW

team02 性能評価報告書

性能評価報告書作成者: 伊藤舜一郎
グループメンバー：
伊藤舜一郎 (学籍番号:1029-32-7548)
植田健斗　(学籍番号:1029-32-6498)

提出期限：6 月 9 日 13 時 15 分　提出日: 2022 年 6 月 3 日

1　性能指標

1.1　回路面積

1.1.1　Flow Summary

作成したプロセッサの Flow Summary は以下の図 1 のようになった。また、中間報告時に完成していた SIMPLE/B の Flow Summary を以下の図 2 で示す。

Flow Summary

<<Filter>>

Flow Status	Successful - Thu May 26 13:49:13 2022
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	processor
Top-level Entity Name	processor
Family	Cyclone IV E
Device	EP4CE30F23I7
Timing Models	Final
Total logic elements	2,548 / 28,848 (9 %)
Total registers	925
Total pins	115 / 329 (35 %)
Total virtual pins	0
Total memory bits	131,072 / 608,256 (22 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	1 / 4 (25 %)

図 1: 作成したプロセッサの Flow Summary

図 1 より、Total logic elements は 2548 個、Total registers は 925 個、Total pins は 115 個、Total memory bits は 131072 ビット、Total PLLs は 1 個と読み取れる。

Flow Summary

<<Filter>>

Flow Status	Successful - Thu May 26 13:46:14 2022
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	processor
Top-level Entity Name	processor
Family	Cyclone IV E
Device	EP4CE30F23I7
Timing Models	Final
Total logic elements	2,129 / 28,848 (7 %)
Total registers	723
Total pins	91 / 329 (28 %)
Total virtual pins	0
Total memory bits	65,536 / 608,256 (11 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

図 2: SIMPLE/B の Flow Summary

図 2 より、Total logic elements は 2129 個、Total registers は 723 個、Total pins は 91 個、Total memory bits は 65536 ビット、Total PLLs は 0 個と読み取れる。

1.1.2 考察

Total logic pins は 2129 から 2548 に増えた理由は、機能拡張をするために fowardingUnit などのパイプライン化に必要な部品や回路の追加したことにより増えたと考えられる。中間報告でのアーキテクチャ検討報告書では、増える logic pins は SIMPLE/B の Total logic pins よりも少なくなるだろうと考察していたが、実際に設計を行行ったところ増加量は 419 であり、SIMPEL/B の Total logic pins よりも少ないため、この考察は正しかったことがわかる。

また、Total registers は 723 から 925 に増えた理由は Total logic pins と同様に、パイプライン化に必要なレジスタを追加したことと、機能拡張のため追加した部品にレジスタが含まれていることが原因だと考えられる。

また、Total pins も 91 から 115 に増えたが、これはクロックサイクル数を記録し 7 セグメント LED に値を出力する Counter の作成に伴い、出力のためのセレクタ信号 8 個と 7 セグメント LED を表示するためのアウトプット信号 8 個を 2 つ用意したためだと考えられる。実際に増加量は 24 であるためこの考察はだいいと考えられる。

また、基数ソートアルゴリズムの実装に伴い、Total memory bits も 2 倍の増えた。また、プロセッサのクロック周波数を 50Hz にするために PLL を追加したため、Total PLLs は 0 から 1 に増えた。

1.2 クロック周波数

1.2.1 CAD での評価値

TimeQuest Timing Analyzer にて Constrains でクロックを設定し、Flow Max Summary を確認する。設定するクロックの名前は clock とする。作成したプロセッサの Flow Max Summary は以下の図 3 のようになった。また、中間報告時に完成していた SIMPLE/B の Flow Max Summary を以下の図 4 で示す。

図 3 より、動作可能な最大周波数は 62.21MHz であると読み取れる。また、図 4 より、動作可能な最大周波数は 70.18MHz であると読み取れる。

Slow 1200mV 100C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	62.21 MHz	62.21 MHz	clock	
2	84.49 MHz	84.49 MHz	altera_reserved_tck	

図 3: 作成したプロセッサの Fmax Summary

Slow 1200mV 100C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	70.18 MHz	70.18 MHz	clock	
2	116.04 MHz	116.04 MHz	altera_reserved_tck	

図 4: SIMPLE/B の Fmax Summary

1.2.2 実機での動作

この FmaxSummary をもとにこのプロセッサが 50MHz で動くようにするために PLL を設定した。この PLL は 40MHz のクロックを 50MHz にすることができる。設定後に、入力した値の数だけフィボナッチ数列の 3 項目から出力する命令 fibonacci と 1024 個の数値を基数ソートする命令 kisuuprot4_r_sorted,kisuuprot4_sorted,kisuuprot4_random をメモリに設定して実機で動作させたところ、問題なく動作した。

また、kisuuprot4_random の命令をメモリに読み込んで実行した場合で、実機で問題なく動作する最大周波数を周波数を増やしながら調べてみたところ、116MHz まで正常に動作し、117MHz 以上だと正常に動作しないことが分かった。

1.2.3 クリティカルパス

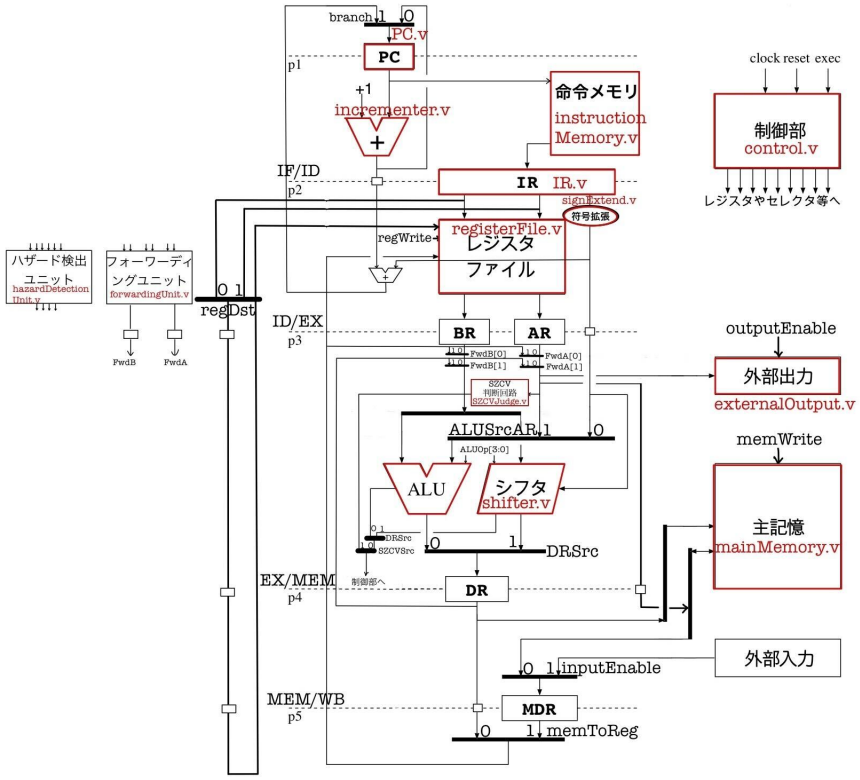
自分たちの設計したプロセッサの回路全体は下図の図 5 のようになった。

TimeQuest Timing Analyzer にて Constrains でクロックを 40MHz に設定し、Slow 1200mV 100C Model での Worst-Case Timing Paths を調べる。Summary of Paths の結果を 500 個出力し、これを Data Delay が大きい順にソートしたところ、下図の図 6 のようになった。

この表において Data Delay の値は 16.033 が最大で、上位 244 個までは緩やかに減少し値は 15.613 となるが、255 個目で Data Delay の値は 5.933 となっておおよそ 1/2.6 になる。これを下図の図 7 で示す。上位 244 個の Path で表れていたのは以下の 8 個のパスだった。

- MEMWBRegister の memToReg から IR,PC
- MDR から IR,PC
- DR_EXMEM から IR,PC
- FwdAFwdB_IDEX から IR,PC
- BR から IR,PC

MEMWRBRegister は制御信号を格納するパイプラインレジスタである。MEMWBRegister から出力される memToReg はマルチプレクサに用いられ、これにより DR_EXMEM もしくは MDR が選択される。MDR は外部入力の値を格納するレジスタである。DR_EXMEM は ALU などの演算結果を格納するレジスタである。FwdAFwdB_IDEX は fowardingUnit の演算結果を格納するレジスタである。BR は IR が保持する命令の Rb/Rd フィールドで指定される汎用レジスタの値を格納するレジスタである。



各レジスタ:register16.v,register4.v,register3.v
各マルチプレクサ:multiplexer16.v,multiplexer4.v,multiplexer3.v
全体:processor.bdf

図 5: プロセッサの回路図

1.2.4 考察

アーキテクチャ検討報告書では、「作成した SIMPLE/B が稼働可能な最大クロック周波数はおよそ 70MHz となった。だが、SIMPLE/B に機能拡張することで遅延時間が延びることを考慮すると、パイプライン化をしたプロセッサの稼働可能な最大周波数は 70MHz よりもさらに下回ると考えられる。」としていたが、作成したプロセッサの Fmax Summary において Fmax は 62.21MHz となり、考察通りになった。

また、Data Delay の最大値は 16.033ns となったが、これを周期とする周波数は 62.37MHz となるので、Fmax の値が 62.21MHz となるのは Data Delay が原因だと考えられる。よって Data Delay の値を小さくすることができればプロセッサの動作可能な最大周波数を上昇させることができると考えられる。Data Delay の値が大きいパスは前述したように以下の 8 個だった。

- MEMWBRegister の memToReg から IR,PC
- FwdAFwdB_IDEX から IR,PC

Slow 1200mV 100C Model

Command Info | Summary of Paths

	Data Delay	Slack	From Node	To Node
1	16.033	23.926	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[0]
2	16.033	23.926	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[1]
3	16.033	23.926	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[2]
4	16.033	23.926	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[3]
5	15.985	23.961	register4:FwdAFwdB_IDEX|q[1]	IR:IR|q[0]
6	15.985	23.961	register4:FwdAFwdB_IDEX|q[1]	IR:IR|q[1]
7	15.985	23.961	register4:FwdAFwdB_IDEX|q[1]	IR:IR|q[2]
8	15.985	23.961	register4:FwdAFwdB_IDEX|q[1]	IR:IR|q[3]
9	15.932	24.027	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[0]
10	15.932	24.027	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[1]
11	15.932	24.027	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[2]
12	15.932	24.027	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[3]
13	15.925	24.021	register4:FwdAFwdB_IDEX|q[1]	IR:IR|q[0]
14	15.925	24.021	register4:FwdAFwdB_IDEX|q[1]	IR:IR|q[1]
15	15.925	24.021	register4:FwdAFwdB_IDEX|q[1]	IR:IR|q[2]
16	15.925	24.021	register4:FwdAFwdB_IDEX|q[1]	IR:IR|q[3]
17	15.909	24.050	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[0]
18	15.909	24.050	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[1]
19	15.909	24.050	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[2]
20	15.909	24.050	MEMWBRegister:inst21|memToReg_WB	IR:IR|q[3]
21	15.903	24.043	register16:BR|q[5]	IR:IR|q[0]
22	15.903	24.043	register16:BR|q[5]	IR:IR|q[1]
23	15.903	24.043	register16:BR|q[5]	IR:IR|q[2]
24	15.903	24.043	register16:BR|q[5]	IR:IR|q[3]
25	15.845	24.104	MEMWBRegister:inst21|memToReg_WB	PC:PC|q[11]

図 6: Sumary of Paths の上位 25 個

Slow 1200mV 100C Model

Command Info | Summary of Paths

	Data Delay	Slack	From Node	To Node
234	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[10]
235	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[9]
236	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[0]
237	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[1]
238	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[2]
239	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[3]
240	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[4]
241	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[5]
242	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[6]
243	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[7]
244	15.613	24.323	register4:FwdAFwdB_IDEX|q[1]	PC:PC|q[8]
245	5.933	13.550	instructionMemory:instructio..._block3a0~porta_address_reg0	IR:IR|q[0]
246	5.893	13.590	instructionMemory:instructio..._block3a0~porta_address_reg0	IR:IR|q[1]
247	5.590	13.891	instructionMemory:instructio..._block3a4~porta_address_reg0	IR:IR|q[15]
248	5.560	13.921	instructionMemory:instructio..._block3a6~porta_address_reg0	IR:IR|q[6]
249	5.512	13.969	instructionMemory:instructio..._block3a4~porta_address_reg0	IR:IR|q[4]
250	5.424	14.024	mainMemory:dataMemory|alts...|ram_block3a0~porta_we_reg	register16:MDR|q[1]
251	5.331	14.152	instructionMemory:instructi...lock3a11~porta_address_reg0	IR:IR|q[11]

図 7: Data Delay の値が約 1/2.6 になるところ

- BR から IR,PC
- DR_EXMEM から IR,PC
- MDR から IR,PC

MEMWBRegister の memToReg と MDR、DR_EXMEM については、IR・PC と次のようにつながっている。制御信号を格納するパイプラインレジスタである MEMWBRegister から出た memToReg はマルチプレクサに用いられ、これにより ALU などの演算結果を格納する DR_EXMEM もしくは外部入力を格納する MDR が選択される。選択された DR_EXMEM,MDR のデータはフォワーディングにより ALU に入力され、これにより SZCV フラグが出力される。このフラグは制御部につながり、branch 信号が生成される。この branch 信号により PC の値がマルチプレクサにより入力される。また、この branch 信号は IR のリセット信号につながっている。

BR は ALU に入力された後、MEMWBRegister の memToReg と MDR と同様のパスをたどる。

FwdAFwdB_IDEX はフォワーディングに用いられ、これによってフォワーディングが行われた後、memToReg と MDR と同様のパスをたどる。

これらのパスの Data Delay の値を小さくする方法としては以下の方法が考えられる。現在の状態だと EX フェーズで多くの部品を通過してデータを伝達する必要があるため、伝達するのに必要なフェーズを 2 つに増やすことで DataDelay を最大で 50%下げることができる。仮に 50%下げることができれば、動作可能な最大周波数がおおよそ 2 倍になるだろうと考えられる。

2 応用プログラムの性能

2.1 作成した応用プログラム

新たに設計した応用プログラムは、提出したファイルの assembly program フォルダに入っている。

応用プログラムとして、レジスタの 1024 番地から 2047 番地までのデータを基数ソートを用いてソートする kisuuprot4.mif を作成した。これは kisuuprot4.txt を simple assembler を用いて mif ファイルに変換したものだ。kisuuprot4 では拡張した命令 ADDI を加えていることと、assembler.py が 0 を受け取ることができなかったため、assembler.py に変更を加えた my_assembler.py というプログラムを使っている。ここで、kisuuprot4 の 1024 から 2047 番地には simple sample の BubbleSort と同じ数列が格納されている。また、kisuuprot4_random.mif,kisuuprot4_sorted.mif,kisuuprot4_r_sorted.mif にはそれぞれソートコンテストで用いる random.mif,sorted.mif,r_sorted.mif の数列が格納されている。また、kisuuprot5 は kisuuprot4 に変更を加えることで、ソートの結果が正しいことを実機でも確認することができるアセンブリ言語プログラムである。

2.1.1 追記

度数ソートを行うプログラムである dosuSort.mif を作成した。dosuSort.txt をアセンブラを用いて変換したものが dosuSort.mif で、ソートコンテストに用いた。これは下位、上位 8 ビットずつ度数ソートを行うことでソートを実行するプログラムである。dosuSort.mif を instructionMemory に、ソートする数列が 1024 から 2047 番地に格納されている mif ファイル (ソートコンテストで用いる random.mif,sort.mif,r_sort.mif) を mainMemory に入れることでソートを行うことができる。

2.2 応用プログラムの設計

2.2.1 my_assembler.py

ソースコード 1: my_assembler.py の変更部分

```
1  def assemble(data):
2      result = []
3      for i in range(len(data)):
4          cmd, args = "", []
5          try:
6              cmd, args = preproc(data[i])
7          except ValueError:
8              print(str(i + 1) + "行目:␣命令の引数が不正です", file=sys.stderr)
9              exit(1)
10         try:
11             if cmd == "ADD":
12         ...
13                 elif cmd == "CONST":
14                     result.append(to_binary(args[0], 16, signed=True))
15                 elif cmd == "ADDI":
16                     result.append(
17                         "10"
18                         + "001"
19                         + to_binary(args[0], 3)
20                         + to_binary(args[1], 8, signed=True)
21                     )
22         ...
```

0 を受け取ることができるように 13、14 行目を追加した。また、ADDI を受け取ることができるように 15〜21 行目を追加した。

2.2.2 kisuuprot4.txt

kisuuprot4.txt に記述したアセンブリ言語を説明する。

ソースコード 2: kisuuprot4 の一部の動作説明

```
1  LI 0, 1
2  SLL 0, 10//r0=1024
3  MOV 6,0
4  ADD 6,0//r6=2048
5  MOV 7,6
6  ADD 7,0//r7=3072
7  LI 1,1//下一桁が 0か 1かでソートする。r1 が基数となる
8  MOV 2,0//r2=1024
9  MOV 3,6//r3=2048
10 LD 4,0(2)
11 LD 5,0(2)
12 AND 4,1//基数をとる
13 BNE 2//r2 に格納されている値の下一桁が 1 なら飛ばす
14 ST 5,0(3)//r3 に r2 に格納されている値を格納
15 ADDI 3,1//r3=r3+1
16 ADDI 2,1//r2=r2+1
17 CMP 2,6
18 BLT −9//r2<2048なら戻る
19 MOV 2,0//r2=1024
20 LD 4,0(2)
21 LD 5,0(2)
22 AND 4,1
23 BE 2//r2 に格納されている値の下一桁が 0 なら飛ばす
24 ST 5,0(3)//r3 に r2 に格納されている値を格納
25 ADDI 3,1//r3=r3+1
```

```
26 ADDI 2,1//r2=r2+1
27 CMP 2,6
28 BLT −9//r2<2048なら戻る
29 LI 1,2//下から 2桁目が 0か 1かでソートする。r1 が基数となる
30 ···
31 BLT −9
32 LD 5, 0(0)
33 OUT 5//1024番地に格納されている数値から順に出力する。
34 LI 1, 1
35 ADD 0, 1
36 CMP 0, 6
37 BLT −6
38 HLT
39 CONST 0
40 ···
41 CONST 0
42 CONST 20908//1024番地。ソートする数列の一番最初
43 ···
44 CONST 32700//2047番地。ソートする数列の一番最後
```

1024 番地から 2047 番地にある数値をはじめに下から 1 桁目が 0 の数値を 2048 番地から並べていき、次に 1 の数値を並べる。その後、並び替えた数列をもとに、下から 2 桁目が 0 の数値を 1024 番地から並べていき、次に 1 の数値を並べる。その後、並び替えた数値をもとに、下から 3 桁目が 0 の数値を 2048 番地から並べていき、次に下一桁が 1 の数値を並べる。... これを 15 桁繰り返し、最後は 1...1,0...0 の順に並べる。これを行うことで基数ソートを実現する。ソートした結果は 1024 番地から 2047 番地まで並んでいる。

2.2.3 kisuuprot5.txt

kisuuprot5 は kisuuport4 のプログラムに変更を加えて、OUT 命令 1 つ次に halt 命令をすることで数値をひとつづつ出力することができる。これによってソートの結果が正しいことを実機でも確認することができる。

2.2.4 dosuSort.txt

dosuSort は下位、上位 8 ビットずつ度数ソートを行うことでソートを実行するプログラムである。以下にプログラムの概要を説明する。ソートする数列の下位 8 ビットの数値に対して、その数値の出現回数とその累積度数を計算する。その後、データを順番に見ていき、データの下位 8 ビットの数値の累積度数-出現回数をアドレスとしてデータを格納する。そして出現回数を-1 する。これを繰り返すことで下位 8 ビットに基づいてソートを行う。これを上位 8 ビットでも行うことで数列全体をソートすることができる。

2.3 応用プログラムの性能の考察

以下ではソートコンテストに用いた kisuuprot4 のプログラムについて性能を評価する。このプログラムではソートする数列がどんな数列だろうと計算量は変わらない。よってこのプログラムは常に計算量が最悪のプログラムであるといえる。このプログラムの実行サイクル数は 16 進数で 4E033 であり、これを 10 進数に変換すると 319539 である。また、実行命令数は 10 進数で 268372 となった。

中間報告では基数ソートを設計する予定がなかったため、ソートコンテストを行うときは BubbleSort を実行するつもりだったが、5 段パイプライン化したとき BubbleSort だと実行サイクル数は 9230000 となる見積もりだった。よってサイクル数は中間報告での見積もりに比べおよそ 1/29 となったといえる。

このプログラムの改善案として 3 つの事が考えられる。

1 つ目は上記のプログラムの 28 行目に当たるところを r3¡r7 とすることだ。これを他の基数でも行えば計算量は少なくなる。だが、すでにソート済みの数列では計算量を減らすことができない。

2 つ目はループ展開をすることだ。ループ展開を行えば条件分岐をする回数を減らすことができるので計算量は少なくなる。だが、すでに kisuuprot4 のソートを行う部分の行数が 374 行あるので、ループ展開を行うにも限界がある。

3 つ目はアルゴリズムを変えることだ。ソートコンテストの結果を見ると、度数 (上位 10bit)+挿入ソートとループ展開を実装した班がソートコンテストでサイクル数 56219 の記録を残しているので、これを自分たちも実装できればサイクル時間はおよそ 1/5.7 になることが予測できる。

2.3.1 追記

実際に度数ソートを行うプログラムを作成したところ、サイクル数は 59428 となった。基数ソートを行うプログラムのサイクル数は 319539 であったため、サイクル数はおよそ 18.6%となった。考察で述べた度数 (上位 10bit)+挿入ソートとループ展開を実装した班のサイクル数よりもサイクル数を小さくすることはできなかったが、ソートプログラムを改善することに成功した。また、実機において 90MHz で dosuSort.mif が動作することを確認した。