2022 年度 3 回生前期学生実験 HW

team02 機能設計仕様書

機能設計仕様書作成者: 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 2 日

1　新たに作成した部品の仕様

1.1　fowadingUnit

1.1.1　回路の仕様

入力・出力

- input [2:0] IDEX_RegRd,EXMEM_RegRd,IFID_RegRs,IFID_RegRd
- input IDEX_RegWrite,EXMEM_RegWrite
- output [1:0] FwdA,FwdB

フォワーディングを行うかどうか判断をするためのユニット。1 ビットの信号 IDEX_RegWrite,EXMEM_RegWrite を受け取り、それが 1 のときでさらに、IFID_RegRs,IFID_RegRd の値が IDEX_RegRd,EXMEM_RegRd と等しいときにフォワーディングを行う。2 ビットの信号 FwdA,FwdB はマルチプレクサに用いられ、FwdA,FwdB の上位 1 ビットが 1 のときは IDEX_RegRd の値が、下位 1 ビットが 1 のときは EXMEM_RegRd の値がフォワーディングされる。

1.1.2　回路の設計

回路の仕様を満たすため、assign 部において、FwdA[1] = (IDEX_RegWrite)&(IDEX_RegRd==IFID_RegRs); とすることで、IDEX_RegWrite が 1 のときかつ IDEX_RegRd と IFID_RegRs の値が等しいとき FwdA の上位 1 ビットが 1 となるようにしている。これと同様の実装を FwdA の下位 1 ビット、FwdB でも行う。fowardingUnit の assign 部は以下のようになる。

ソースコード 1: fowardingUnit の assign 文

```
1 assign FwdA[1] = (IDEX_RegWrite)&(IDEX_RegRd==IFID_RegRs);
2 assign FwdA[0] = (EXMEM_RegWrite)&(EXMEM_RegRd==IFID_RegRs);
3 assign FwdB[1] = (IDEX_RegWrite)&(IDEX_RegRd==IFID_RegRd);
4 assign FwdB[0] = (EXMEM_RegWrite)&(EXMEM_RegRd==IFID_RegRd);
```

1.2　clockCounter

このモジュールは processor フォルダ内にある。

1.2.1　回路の仕様

入力・出力

- input [15:0] instruction,
- input systemRunning,clock,reset,
- output reg [7:0] SEG_X,SEG_Y,
- output reg [3:0] selectX,selectY

このモジュールはソートコンテストにおいてサイクル数を数えて 7 セグメント LED に出力するためのものである。1 ビットの systemRunning を受け取ってそれが 1 のときにカウントを開始して、16 ビットの命令を受け取ってそれが halt 命令でないならカウントを 1 足し続け、halt 命令ならカウントを 4 足してカウントをやめる。7 セグメント LED への出力は拡張ボード上で行わず、MU500-RXSET 上で行う。そのためにセレクタ信号 selectX,selectY と 7 セグメント LED を点灯させるための 8 ビットの信号 SEG_X,SEG_Y を出力するようにした。

1.2.2 回路の設計

下の図 1 の X の部分に出力の selectX,SEG_X が、Y の部分に出力の selectY,SEG_Y が対応するように設計、ピンアサインメントを行う。また、点灯のさせ方は externalOutput と同様に行う。

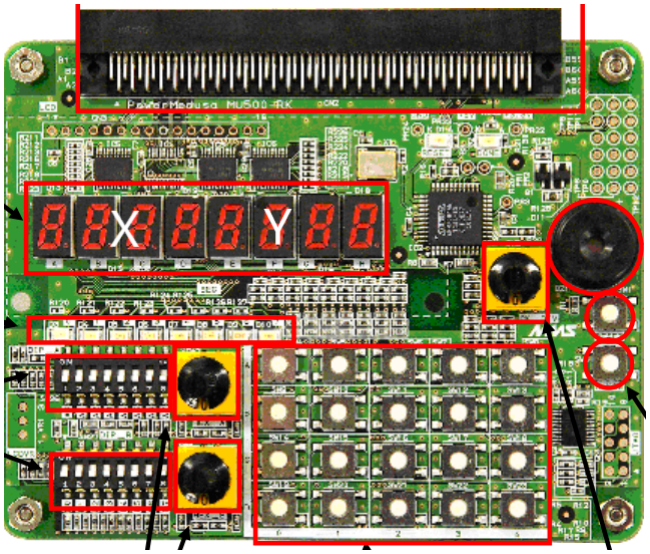

図 1: 点灯させる 7 セグメント LED

また、externalOutput と同様に、outFunc 関数は 4 ビットの数値を受け取って、その値に応じて 16 進数の値として 7SEGLED が点灯するように 8 ビットの数値を返す。

ソースコード 2: outFunc

```
1  function [7:0] outFunc;
2          input [3:0] a;
3          begin
4                  case (a)
5                  4'b0000:outFunc = 8'b1111_1100;
6                  4'b0001:outFunc = 8'b0110_0000;
7                  4'b0010:outFunc = 8'b1101_1010;
8                  4'b0011:outFunc = 8'b1111_0010;
9                  4'b0100:outFunc = 8'b0110_0110;
10                 4'b0101:outFunc = 8'b1011_0110;
11                 4'b0110:outFunc = 8'b1011_1110;
12                 4'b0111:outFunc = 8'b1110_0000;
13                 4'b1000:outFunc = 8'b1111_1110;
14                 4'b1001:outFunc = 8'b1111_0110;
15                 4'b1010:outFunc = 8'b1110_1110;
16                 4'b1011:outFunc = 8'b0011_1110;
17                 4'b1100:outFunc = 8'b0001_1010;
18                 4'b1101:outFunc = 8'b0111_1010;
```

```
19                  4'b1110:outFunc = 8'b1001_1110;
20                  4'b1111:outFunc = 8'b1000_1110;
21                  default:outFunc = 8'b0000_0000;
22 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣endcase
23 ␣␣␣␣␣␣␣␣end
24 endfunction
```

また、always 部は以下のようになっている。

ソースコード 3: always 部

```
1  reg [31:0] counter;
2  reg start;
3  reg stop;
4  reg [2:0] pattern1,pattern2;
5
6  always @(posedge clock) begin
7          if(reset) begin
8                  counter <= 0;
9                  start <= 1'b0;
10 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣stop ␣<= ␣1'b0;
11         end else begin
12                 if(systemRunning==1'b1&~start) ␣begin
13 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣start ␣<= ␣1'b1;
14                         counter <= counter + 1;
15                 end else if(({instruction[15:14],instruction[7:4]}==6'b11_1111)&~stop) ␣begin
16 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣stop ␣<= ␣1'b1;
17                         counter <= counter + 4;
18                 end else if(start&~stop) begin
19                         counter <= counter + 1;
20                 end
21         end
22
23         if (reset) begin
24                 selectX <= 4'b0000;
25 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣pattern1 ␣<= ␣3'b000;
26                 SEG_X <= outFunc(counter[19:16]);
27         end else if( pattern1 == 3'b000) ␣begin
28 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectX ␣<= ␣4'b1110;
29                 pattern1 <= pattern1 + 1;
30         end else if( pattern1 == 3'b001) ␣begin
31 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectX ␣<= ␣4'b1111;
32                 SEG_X <= outFunc(counter[31:28]);
33                 pattern1 <= pattern1 + 1;
34         end else if( pattern1 == 3'b010) ␣begin
35 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectX ␣<= ␣4'b1101;
36                 pattern1 <= pattern1 + 1;
37         end else if( pattern1 == 3'b011) ␣begin
38 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectX ␣<= ␣4'b1111;
39                 SEG_X <= outFunc(counter[27:24]);
40                 pattern1 <= pattern1 + 1;
41         end else if( pattern1 == 3'b100) ␣begin
42 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectX ␣<= ␣4'b1011;
43                 pattern1 <= pattern1 + 1;
44         end else if( pattern1 == 3'b101) ␣begin
45 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectX ␣<= ␣4'b1111;
46                 SEG_X <= outFunc(counter[23:20]);
47                 pattern1 <= pattern1 + 1;
48         end else if( pattern1 == 3'b110) ␣begin
49 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectX ␣<= ␣4'b0111;
50                 pattern1 <= pattern1 + 1;
51         end else if( pattern1 == 3'b111) ␣begin
52 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectX ␣<= ␣4'b1111;
53                 SEG_X <= outFunc(counter[19:16]);
54                 pattern1 <= pattern1 + 1;
55         end
56
57         if (reset) begin
58                 selectY <= 4'b0000;
```

```
59 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣pattern2␣<=␣3'b000;
60                     SEG_Y <= outFunc(counter[3:0]);
61             end else if( pattern2 == 3'b000)␣begin
62 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectY␣<=␣4'b1110;
63                     pattern2 <= pattern2 + 1;
64             end else if( pattern2 == 3'b001)␣begin
65 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectY␣<=␣4'b1111;
66                     SEG_Y <= outFunc(counter[15:12]);
67                     pattern2 <= pattern2 + 1;
68             end else if( pattern2 == 3'b010)␣begin
69 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectY␣<=␣4'b1101;
70                     pattern2 <= pattern2 + 1;
71             end else if( pattern2 == 3'b011)␣begin
72 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectY␣<=␣4'b1111;
73                     SEG_Y <= outFunc(counter[11:8]);
74                     pattern2 <= pattern2 + 1;
75             end else if( pattern2 == 3'b100)␣begin
76 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectY␣<=␣4'b1011;
77                     pattern2 <= pattern2 + 1;
78             end else if( pattern2 == 3'b101)␣begin
79 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectY␣<=␣4'b1111;
80                     SEG_Y <= outFunc(counter[7:4]);
81                     pattern2 <= pattern2 + 1;
82             end else if( pattern2 == 3'b110)␣begin
83 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectY␣<=␣4'b0111;
84                     pattern2 <= pattern2 + 1;
85             end else if( pattern2 == 3'b111)␣begin
86 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣selectY␣<=␣4'b1111;
87                     SEG_Y <= outFunc(counter[3:0]);
88                     pattern2 <= pattern2 + 1;
89             end
90
91 end
```

counter はクロック数を数えるためのカウンタである。7〜21 行目で 1 ビットの systemRunning を受け取ってそれが 1 のときにカウントを開始して、16 ビットの命令を受け取ってそれが halt 命令でないならカウントを 1 足し続け、halt 命令ならカウントを 4 足してカウントをやめるように実装している。クロックが立ち上がったときに、reset 信号を受け取ったらカウンタやその他のレジスタを初期状態に戻す。systemRunnning が 1 になったら start を 1 にしてカウンタを 1 足しはじめ、start が 1 かつ stop が 0 のときはカウンタに 1 を足す。stop が 0 のときに halt 命令を受け取ったら stop を 1 にしてカウンタに 4 を足して、カウンタの増加をやめる。

また、23〜55 行目で X 部の LED の点灯を行っている。点灯の仕組みは以下のようになっている。

SEG_X にカウンタの上位 1〜4 ビットの値を格納→ 1 クロック後に X の上から 1 桁目が点灯するようにセレクタ信号を更新

↓

1 クロック後にセレクタ信号を 1111 にして何も点灯しない。SEG_X にカウンタの上位 5〜8 ビットの値を格納→ 1 クロック後に X の上から 2 桁目が点灯するようにセレクタ信号を更新

↓

1 クロック後にセレクタ信号を 1111 にして何も点灯しない...

これをループすることであたかも同時に各 LED が点灯して見えるようにしている。また、Y 部でも同様に 57〜89 行目で実装している。

2　性能評価

2.1　fowardingUnit

2.1.1　回路規模

fowardingUnit の回路規模は下図の図 2 のようになった。ここから Total logic elements は 8 で、Total pins は 18 であることが読み取れる。

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Jun 02 04:46:20 2022
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Editio
Revision Name	forwardingUnit
Top-level Entity Name	forwardingUnit
Family	Cyclone IV E
Device	EP4CE30F23I7
Timing Models	Final
Total logic elements	8 / 28,848 (< 1 %)
Total registers	0
Total pins	18 / 329 (5 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

図 2: fowardingUnit の Flow Summary

2.1.2　考察

はじめは、if 文を用いて設計を行おうとしていたが、その場合モジュールを作成するとマルチプレクサを用いたような回路が設計されてしまい、この設計より Total logic elements は多くなるだろうと予測されるので、この設計は最適化されているのではないのだろうかと考えた。また、組み合わせ回路のため CAD での予想遅延時間は与えられなかったが、if 文を用いた設計より個の設計は遅延時間もすくないだろうと予測される。

2.2　clockCounter

2.2.1　回路規模

clockCounter の回路規模は下図の図 3 のようになった。ここから Total logic elements は 197 で、Total pins は 43、Total registers は 55 であることが読み取れる。

2.2.2　周波数

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 として 100MHz とした。作成した clockCounter の Flow Max Summary は以下の図 4 のようになった。ここからこの回路が動作可能な最大周波数は 196.81MHz であることがわかる。

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Jun 02 05:00:39 2022
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	clockCounter
Top-level Entity Name	clockCounter
Family	Cyclone IV E
Device	EP4CE30F23I7
Timing Models	Final
Total logic elements	197 / 28,848 (< 1 %)
Total registers	55
Total pins	43 / 329 (13 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

図 3: clock の Flow Summary

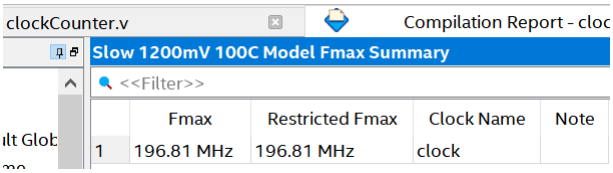

図 4: clock の FMax Summary

2.2.3 クリティカルパス

また、Slow 1200mV 100C Model での Worst-Case Timing Paths を調べる。Summary of Paths の結果を 300 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 5 のようになった。図 5 からクリティカルパスは clock の値の足し算で発生していることがわかる。

Slow 1200mV 100C Model

Command Info | Summary of Paths

	Slack	Data Delay	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew
1	4.919	5.026	counter[0]	counter[29]	clock	clock	10.000	-0.073
2	5.103	4.839	counter[3]	counter[29]	clock	clock	10.000	-0.076
3	5.135	4.812	counter[0]	counter[25]	clock	clock	10.000	-0.071
4	5.172	4.773	counter[0]	counter[31]	clock	clock	10.000	-0.073
5	5.186	4.758	counter[1]	counter[29]	clock	clock	10.000	-0.074
6	5.215	4.732	counter[0]	counter[30]	clock	clock	10.000	-0.071
7	5.233	4.709	counter[5]	counter[29]	clock	clock	10.000	-0.076
8	5.235	4.709	counter[3]	counter[30]	clock	clock	10.000	-0.074
9	5.252	4.690	counter[2]	counter[29]	clock	clock	10.000	-0.076
10	5.283	4.662	counter[0]	counter[27]	clock	clock	10.000	-0.073

図 5: clock のクリティカルパス

2.2.4　考察

性能評価報告書で述べたが、作成したプロセッサの動作可能な最大周波数は 60MHz 程度であるため、clockCounter の動作可能な最大周波数はプロセッサの動作可能な最大周波数に比べ非常に大きい。また、プロセッサのクリティカルパスに clockCounter の部分は含まれていなかったため、clockCounter を改善してもプロセッサの機能の向上は見込めないのではないのだろうかと自分は考えた。

また、clockCounter は各 7 セグメント LED の表示の切り替えに clock を用いているために、clock の値が大きくなると同時に表示されているように見えるが、周期も短くなるため表示が暗くなる。拡張ボードでクロック数を表示すれば明るさの問題は解決するが、externalOutput の仕様やピンアサインメントを変更する必要があり面倒である。拡張ボードを使用せずにこの問題を解決する方法は何かないだろうかと自分は考えたが思いつかなかった。

3　実験の感想

計算機科学概論の講義でアセンブリ言語について学び、計算機の構成・計算機アーキテクチャの講義でプロセッサの仕組みについて説明を受けていたが、講義だけでは理解が進んでいなかった。今回、ハードウェア実験を通してプロセッサの作成、アセンブリプロセッサの作成を行うことでこれらの理解が深まった。また、他の人と実装を行う際にはコミュニケーションや認識のすり合わせがいかに大切であるかを学ぶことができたので良かった。