ネットワーク系演習計算機ハードウェア最終レポート

南田宗太郎 (28114124)

2018 年 7 月 25 日

1　レポート課題 7

1.1　7-1 の課題について

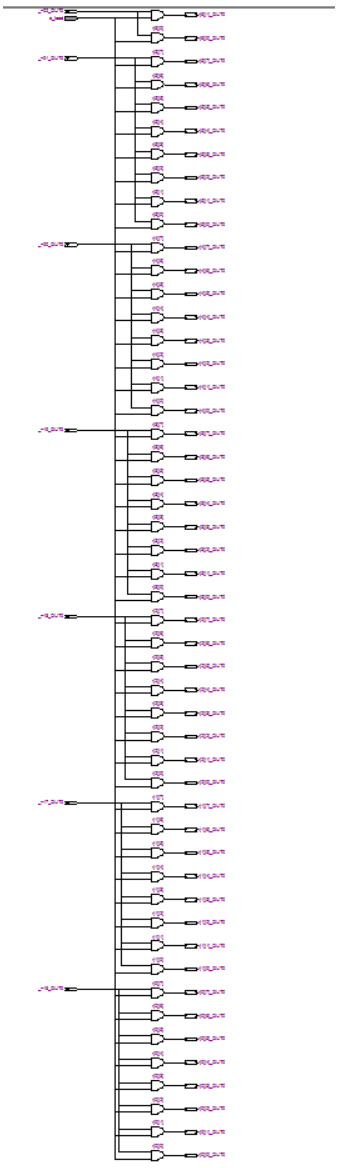
八個の 8 ビットレジスタを持つように拡張したソースコードを以下に記述する。

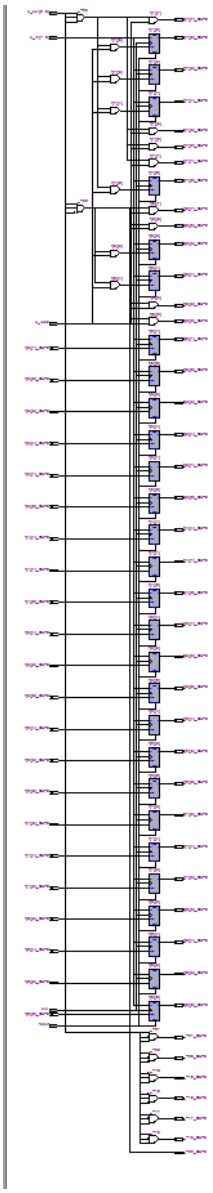
```
SUBDESIGN reg_d
(
   a_sel[2..0], b_sel[2..0], c_sel[2..0], c_load : INPUT;
   a_out[7..0], b_out[7..0] : OUTPUT;
   c_in[7..0],clk,reset : INPUT;
)

VARIABLE
r[7..0][7..0] : DFFE;

BEGIN
r[][].(clk,clrn) = (clk,!reset);
CASE a_sel[] IS
WHEN 0 => a_out[] = r[0][].q;
WHEN 1 => a_out[] = r[1][].q;
WHEN 2 => a_out[] = r[2][].q;
WHEN 3 => a_out[] = r[3][].q;
WHEN 4 => a_out[] = r[4][].q;
WHEN 5 => a_out[] = r[5][].q;
WHEN 6 => a_out[] = r[6][].q;
```

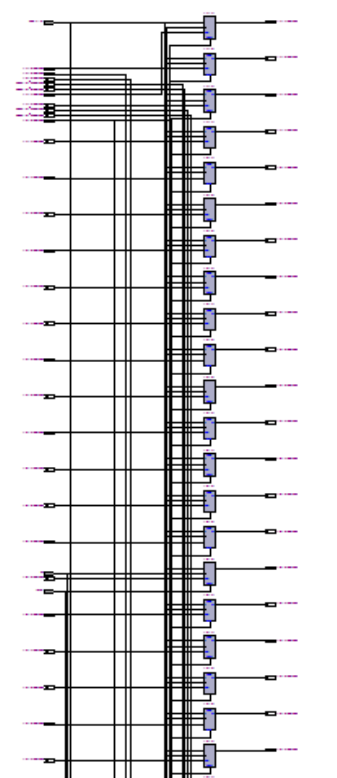
```
WHEN 7 => a_out[] = r[7][].q;
END CASE;
CASE b_sel[] IS
WHEN 0 => b_out[] = r[0][].q;
WHEN 1 => b_out[] = r[1][].q;
WHEN 2 => b_out[] = r[2][].q;
WHEN 3 => b_out[] = r[3][].q;
WHEN 4 => b_out[] = r[4][].q;
WHEN 5 => b_out[] = r[5][].q;
WHEN 6 => b_out[] = r[6][].q;
WHEN 7 => b_out[] = r[7][].q;
END CASE;

FOR i IN 0 TO 7 GENERATE
r[i][].d = c_in[];
END GENERATE;

CASE c_sel[] IS
WHEN 0 => r[0][].ena = c_load;
WHEN 1 => r[1][].ena = c_load;
WHEN 2 => r[2][].ena = c_load;
WHEN 3 => r[3][].ena = c_load;
WHEN 4 => r[4][].ena = c_load;
WHEN 5 => r[5][].ena = c_load;
WHEN 6 => r[6][].ena = c_load;
WHEN 7 => r[7][].ena = c_load;
END CASE;
END;
```

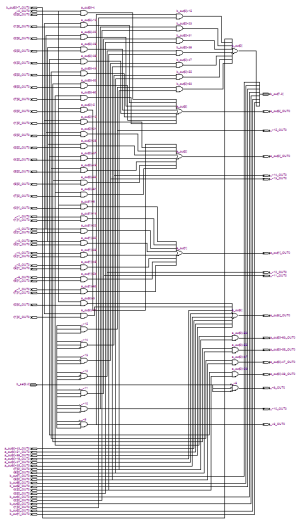
1.2　7-2 の課題について

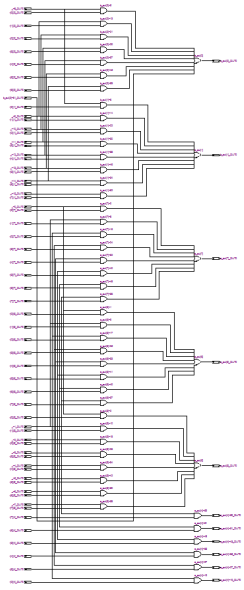
ブロック図を以下に添付する。

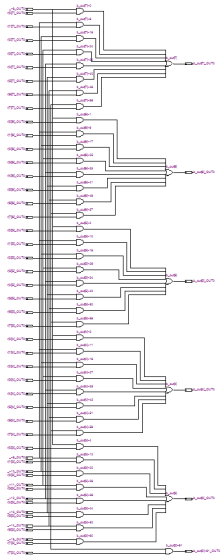












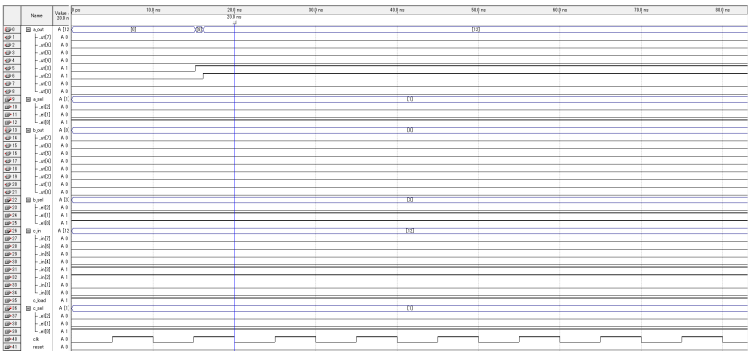
1.3　7-3 の課題について

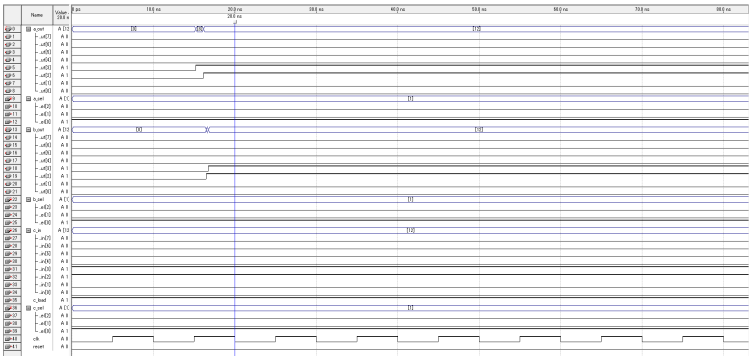
以下にシュミレーションの結果の波形図を添付する。

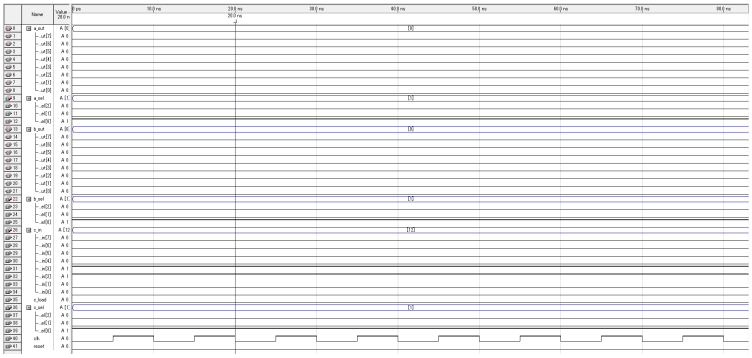
波形図 1 では c_sel に 1,b_sel に 3,a_sel に 1,c_in に 12 を入力している。今回 c_sel で 1 を選んだのでレジスタ番号 1 のレジスタに 12 が入力され、a_sel に 1 が入力されているため a_out にはレジスタ番号 1 に格納されている 12 が出力され b_sel に 3 が入力されているためレジスタ番号 3 に格納されている 0 が出力されることが期待される結果である。波形図 1 より期待される結果が正しく出力されていることがわかる。

波形図 1 では c_sel に 1,b_sel に 3,a_sel に 1,c_in に 12 を入力している。波形図 1 とは違い a_sel,b_sel どちらとも 1 が入力されているため、a_out,b_out どちらも出力されるものは 12 が期待

される結果である。波形図 2 より期待される結果が正しく出力されていることがわかる。
波形図 3 は c_load の入力を 0 にしたものである。c_load が 0 なので a_out,b_out どちらの出力も 0 が期待される結果である。波形図 3 より期待される結果が正しく出力されていることがわかる。







2　レポート課題 8

2.1　8-1

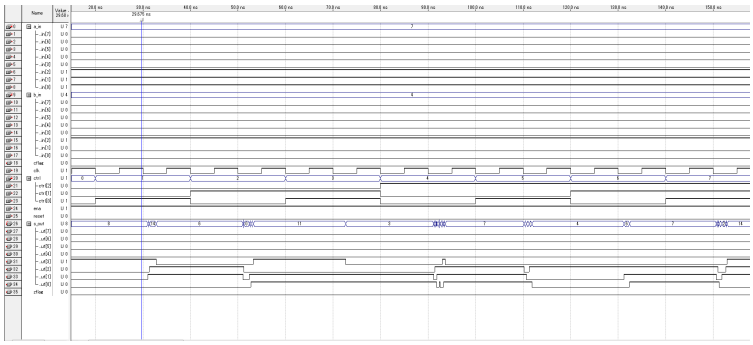
　alu_d を改造し,8 つの機能に拡張しました。，具体的には既存の 4 つの機能に加えて大小比較、ビット毎の論理積、ビット毎の論理和，ビットシフトを追加しました。

```
SUBDESIGN alu_d
(
 clk, reset, ena : INPUT;
 a_in[7..0], b_in[7..0] : INPUT;
 ctrl[1..0] : INPUT;
 s_out[7..0] : OUTPUT;
 cflag, zflag : OUTPUT;
)

VARIABLE
cff, zff : DFFE;

BEGIN
cff.(clk, clrn, ena) = (clk, !reset, ena);
zff.(clk, clrn, ena) = (clk, !reset, ena);
cflag = cff.q;
zflag = zff.q;
zff.d = (s_out[] == 0);

CASE ctrl[] IS
WHEN 0 => (cff.d, s_out[]) = (B"0", a_in[]+1);
WHEN 1 => (cff.d, s_out[]) = (B"0", a_in[]-1);
WHEN 2 => (cff.d, s_out[]) = (B"0", a_in[])+(B"0", b_in[]);
WHEN 3 => (cff.d, s_out[]) = (B"0", a_in[])-(B"0", b_in[]);
WHEN 4 => IF a_in[] >= b_in[]
THEN s_out[] = a_in[];
ELSE s_out[] = b_in[];
END IF;
WHEN 5 => FOR i IN 0 TO 7 GENERATE
s_out[i] = a_in[i]&b_in[i];
```

```
END GENERATE;
WHEN 6 => FOR i IN 0 TO 7 GENERATE
s_out[i] = a_in[i]#b_in[i];
END GENERATE;
WHEN 7 => (s_out[]) = (a_in[6..0],B"0");

END CASE;
END;
```

2.2 8-2

以下にシュミレーションをした際の波形図を添付する。クロックの 2 周期を境に ctrl を 1 ずつ増やしていくように入力を定めた。ctrl が 0 の時、a_in の値が 1 増えたものが s_out に出力されている。これは期待していた正しい結果である。
ctrl が 1〜3 の時も期待していた結果が出力されている。
ctrl が 4 の時、期待される出力は a_in と b_in を大小比較して大きい方の値である a_in が出力されることである。波形図より期待通りの出力が得られているので正しいと言える。
ctrl が 5 の時、期待される出力は a_in と b_in の論理積が出力されることである。具体的には 00000111 と 00000100 の論理積なので 4 が出力されることが期待されている。波形図より期待通りの出力が得られているので正しいと言える。
ctrl が 6 の時、期待される出力は a_in と b_in の論理和が出力されることである。具体的には 00000111 と 00000100 の論理和なので 7 が出力されることが期待されている。波形図より期待通りの出力が得られているので正しいと言える。
ctrl が 7 の時、期待される出力は a_in の 1 ビットだけ右に論理シフトした値が出力されることである。具体的には 00000111 の右ビットシフトなので 14 が出力されることが期待されている。波形図より期待通りの出力が得られているので正しいと言える。



3　レポート課題 9

3.1　9-1

私は load,store,add 命令を利用して $f(n) = f(n-1) + f(n-2)$ のフィボナッチ数列を計算するプログラムを組みました。プログラムの動きと正しいことを報告します。
fetcha でメモリから命令部分をとって来てそれを ira に格納するようにしました。fetchb でメモリからデータ部分をとって来てそれを irb に格納するようにしました。execa で ira に入っている命令を実行するようにしました。つまり今回はステージの更新と計算の実行は別のタイミングで行う必要があるのでステージの管理をしているフリップフロップはネガティブエッジトリガになるように、そのほかはポジティブエッジトリガになるように接続しました。波形を見るとそれが正しく動いていることがわかります。次にメモリの初期値について報告します。フィボナッチ数列の初項と２項目を 120 と 128 番地に格納しています。０番地と２番地の命令は load 命令でレジスタに格納するようにしています。４番地の命令は add 命令で結果をレジスタに格納しています。６番地の命令は store 命令で先ほどの結果を格納したものをメモリの 136 番地に書き込むようにしています。８番地の命令は先ほど格納して第２項目と計算した第３項目が格納されているレジスタを指定して add 命令を実行しています。１０番地はその結果を 144 番地に書き込むようにしています。以降、同じ動きです。そして今回第６項までしか求められませんでした。原因は実行時間が足りなくこれ以上にメモリに命令を記述すると中途半端なところで実行が終わりハザードを起こす危険性があったのでこれ以上は求められませんでした。

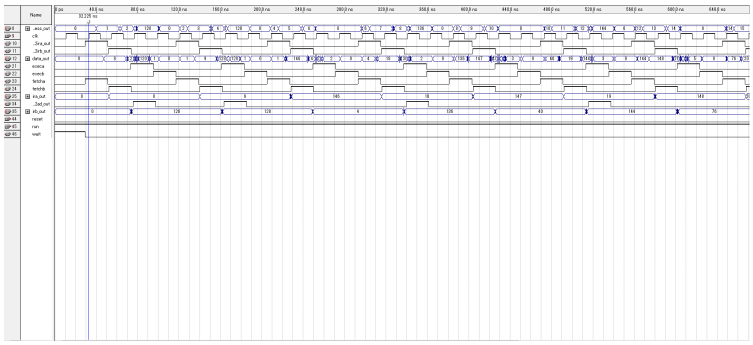


図１　波形図（途中まで）

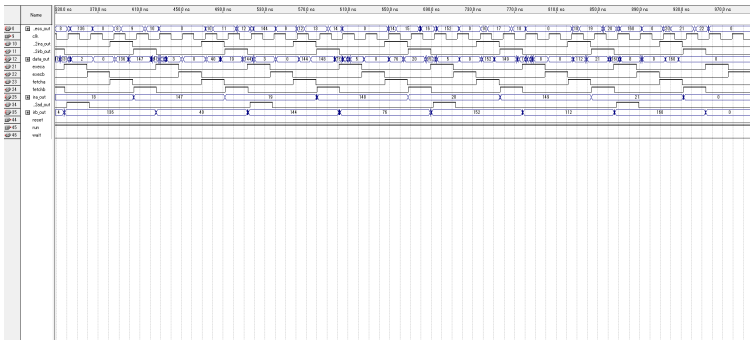


図 2　波形図（前の続き）

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	8	120	9	128	146	4	18	136
8	147	40	19	144	148	76	20	152
16	149	112	21	160	0	0	0	0
24	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0
56	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0
72	0	0	0	0	0	0	0	0
80	100	0	0	0	0	0	0	0
88	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0
104	0	0	0	0	0	0	0	0
112	0	0	0	0	0	0	0	0
120	1	0	0	0	0	0	0	0
128	1	0	0	0	0	0	0	0
136	2	0	0	0	0	0	0	0
144	3	0	0	0	0	0	0	0
152	5	0	0	0	0	0	0	0
160	8	0	0	0	0	0	0	0
168	0	0	0	0	0	0	0	0
176	0	0	0	0	0	0	0	0
184	0	0	0	0	0	0	0	0
192	0	0	0	0	0	0	0	0
200	0	0	0	0	0	0	0	0
208	0	0	0	0	0	0	0	0
216	0	0	0	0	0	0	0	0
224	0	0	0	0	0	0	0	0
232	0	0	0	0	0	0	0	0
240	0	0	0	0	0	0	0	0
248	0	0	0	0	0	0	0	0

図 3　メモリの結果（0 19 番地が初期値、120 番地より縦に実行結果）

3.2　9-2

先ほどのフィボナッチ数列をシフトを用いて拡張して $f(n) = f(n-1) + 2f(n-2)$ のフィボナッチ数列を計算するプログラムを組みました。先ほどとの変更点を報告します。先ほどのメモリに初期値と違い load してからそのまま add するのではなく、最初に load した方を shift 命令を使い二倍してから add するように初期値を書き換えました。以下に波形図とメモリの結果を添付します。

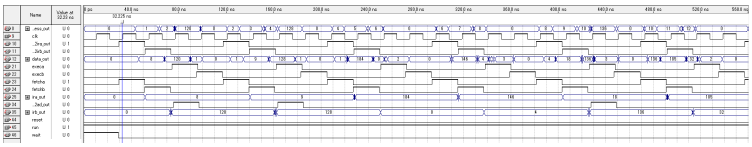


図 4　波形図（途中まで）

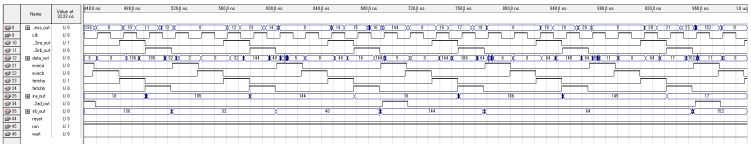


図 5　波形図（前の続き）

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	8	120	9	128	184	0	146	4
8	18	136	185	32	144	40	16	144
16	186	64	145	64	17	152	0	0
24	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0
56	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0
72	0	0	0	0	0	0	0	0
80	100	0	0	0	0	0	0	0
88	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0
104	0	0	0	0	0	0	0	0
112	0	0	0	0	0	0	0	0
120	1	0	0	0	0	0	0	0
128	1	0	0	0	0	0	0	0
136	3	0	0	0	0	0	0	0
144	5	0	0	0	0	0	0	0
152	11	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0
168	0	0	0	0	0	0	0	0
176	0	0	0	0	0	0	0	0
184	0	0	0	0	0	0	0	0
192	0	0	0	0	0	0	0	0
200	0	0	0	0	0	0	0	0
208	0	0	0	0	0	0	0	0
216	0	0	0	0	0	0	0	0
224	0	0	0	0	0	0	0	0
232	0	0	0	0	0	0	0	0
240	0	0	0	0	0	0	0	0
248	0	0	0	0	0	0	0	0

図 6　メモリの結果（0 21 番地が初期値、120 番地より縦に実行結果）