情報知能工学実験 II
CAD とシミュレーション

神戸大学 工学部 情報知能工学科

1　実験方法

1.1　基本ゲートの記述とシミュレーション

下記の基本ゲートについて，以下の操作を行ない，入力として，`(in1 in2) = (0 0), (1 0), (0 1), (1 1)` を与えた時の結果を検証して，真理値表およびタイミングチャートを作成せよ．

```
+++++++++++++++++++++++++++++++++++++++++++++++
media00% seconds          SECONDS の立ち上げ
SFL> sflread {and, or, nand, nor, eor, enor}.sfl
                          SFL ソースの読み込み
SFL> gate.sim {and, or, nand, nor, eor, enor}
+++++++++++++++++++++++++++++++++++++++++++++++
```

ただし，シミュレーションを行なう時のスクリプトファイルは以下に与える．なお，ファイル `f` を `forward +1` ファイル `p` を`print "%d\n" $1` として，簡略化している．

```
"gate.sim"
==================
install $1 /
p o1
set /i1 0
set /i2 0
hold /i1
hold /i2
f
p o1
set /i1 0
set /i2 1
hold /i1
hold /i2
f
p o1
set /i1 1
set /i2 0
hold /i1
hold /i2
f
p o1
set /i1 1
set /i2 1
hold /i1
hold /i2
f
p o1
==================
```

1.1.1　AND ゲート

```
############################
module and {
    input     i1;
    input     i2;
    output    o1;
    o1 = i1 & i2;
}
############################
```

1.1.2　OR ゲート

```
############################
module or {
    input     i1;
    input     i2;
    output    o1;
    o1 = i1 | i2;
}
############################
```

1.1.3　NAND ゲート

```
############################
module nand {
    input     i1;
    input     i2;
    output    o1;
    o1 = ^(i1 & i2);
}
############################
```

1.1.4　NOR ゲート

```
############################
module nor {
    input     i1;
    input     i2;
    output    o1;
    o1 = ^(i1 | i2);
}
############################
```

1.1.5　XOR ゲート

```
############################
module eor {
    input     i1;
    input     i2;
    output    o1;
    o1 = i1 @ i2;
}
############################
```

1.1.6　XNOR ゲート

```
############################
module enor {
    input     i1;
    input     i2;
    output    o1;
    o1 = ^(i1 @ i2);
}
############################
```

1.2　フリップフロップ

以下操作にしたがい，のＤフリップフロップの動作をシミュレートし，動作を確認せよ．

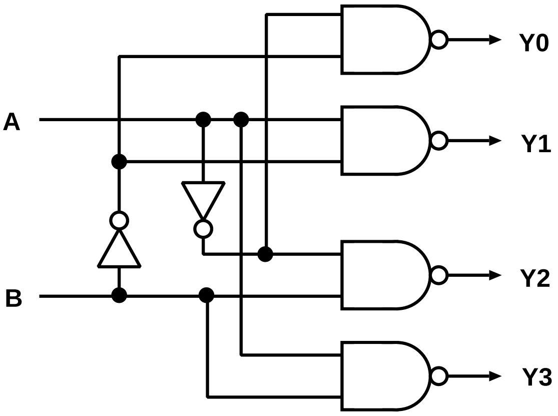
```
++++++++++++++++++++++++++++
media00% seconds
SFL> sflread dff.sfl
SFL> dff.sim
++++++++++++++++++++++++++++
############################
module dff {
    input     in;
    output    out;
    reg       a;
    par {
       a := in;
       out = a;
    }
}
############################
```

```
"dff.sim"
============================
install dff /
set /in 1
hold /in
p out
f
p out
set /in 0
hold /in
p out
f
p out
set /in 0
hold /in
p out
f
p out
set /in 1
hold /in
p out
f
p out
============================
```

［課題１］
このファイルを元に，Ｔ（トグルフリップフロップ）および，ＪＫフリップフロップの動作を記述する SFL のソースを示せ．

1.3　デコーダ回路

以下の操作にしたがい，下図に示す 2-4 デコーダの動作を確認せよ．



```
++++++++++++++++++++++++++++
media% seconds
SFL> sflread decoder.sfl
SFL> install decoder /
SFL> set in ??      ２進数の入力
SFL> hold in
SFL> forward +1
SFL> print "%b\n", out
++++++++++++++++++++++++++++
```

なお，ファイル pb を

```
print "%b\n" $1
```

として，上記操作を簡略化することができる．

```
#######################################
module decoder {
    input     in<2>;
    output    out<4>;
    tmp       cnv<4>;
    par {
      any {
        ^in<0> & ^in<1> : cnv = 0b1110;
         in<0> & ^in<1> : cnv = 0b1101;
        ^in<0> &  in<1> : cnv = 0b1011;
         in<0> &  in<1> : cnv = 0b0111;
      }
    out = cnv;
    }
}
#######################################
```

1.4　７セグメントＬＥＤのデコーダ

以下に，７セグメントＬＥＤを表示するためのデコーダ回路を示す．スクリプトにしたがってシミュレーションを行ない動作を確認せよ．

```
++++++++++++++++++++++++++++++
media% seconds
SFL> sflread seven_seg.sfl
SFL> install seven_seg /
SFL> test7seg.sim ????
++++++++++++++++++++++++++++++
########################################
module seven_seg {
    input   in<4>;
    output  out<7>;
    tmp     cnv<7>;
    par {
        any {
            in == 0x0 : cnv = 0b1111110;
            in == 0x1 : cnv = 0b1100000;
            in == 0x2 : cnv = 0b1011011;
            in == 0x3 : cnv = 0b1110011;
            in == 0x4 : cnv = 0b1100101;
            in == 0x5 : cnv = 0b0110111;
            in == 0x6 : cnv = 0b0111111;
            in == 0x7 : cnv = 0b1100010;
            in == 0x8 : cnv = 0b1111111;
            in == 0x9 : cnv = 0b1110111;
            in == 0xa : cnv = 0b1101111;
            in == 0xb : cnv = 0b0111101;
            in == 0xc : cnv = 0b0011001;
            in == 0xd : cnv = 0b1111001;
            in == 0xe : cnv = 0b0011111;
            in == 0xf : cnv = 0b0001111;
        }
        out = cnv;
    }
}
########################################
"test7seg.sim"
========================================
set in $1
hold in
f
speak 7seg.out
pb out
flush
========================================
```

７セグメントのＬＥＤの動作を示すプログラム 7seg.c を以下に示す．

```
#include <stdio.h>
main()
{
  FILE *fp;
  int seven_out = 0;
  char in[10];
  int a, b, c, d, e, f, g;
  a = b = c = d = e = f = g = 0;
  while(1){
    system("touch 7seg.out");
    fp = fopen("7seg.out", "r");
    while (fgets(in, sizeof(in), fp) != NULL);
    fclose(fp);
    seven_out = (in[0]-'0')*64 +
                (in[1]-'0')*32 +
                (in[2]-'0')*16 +
                (in[3]-'0')*8  +
                (in[4]-'0')*4  +
```

```
                (in[5]-'0')*2  +
                (in[6]-'0')*1 ;
    a = seven_out & 0x40;
    b = seven_out & 0x20;
    c = seven_out & 0x10;
    d = seven_out & 0x08;
    e = seven_out & 0x04;
    f = seven_out & 0x02;
    g = seven_out & 0x01;
    system("clear");
    if(f) printf("      ====\n");
    if(!f) printf("         \n");
    if(a && e) printf("    //   //\n");
    if(a && !e) printf("        //\n");
    if(!a && e) printf("    //    \n");
    if(!a && !e) printf("         \n");
    if(g) printf("    ====\n");
    if(!g) printf("       \n");
    if(b && d) printf("  //   //\n");
    if(b && !d) printf("       //\n");
    if(!b && d) printf("  //    \n");
    if(!b && !d) printf("       \n");
    if(c) printf("  ====\n");
    if(!c)  printf("     \n");
   sleep(1);
   }
}
```

1.5　カウンタ回路

以下に，4ビットのカウンタ回路を示す．スクリプトにしたがってシミュレーションを行ない動作を確認せよ．

```
++++++++++++++++++++++++++
media% seconds
SFL> sflread counter.sfl
SFL> install counter /
SFL> set start 1
SFL> speak 7seg.out
SFL> c.sim
++++++++++++++++++++++++++
"c.sim"
==========================
f
pb disp
flush
==========================
```

```
##############################################
  /*******************************/
  /* 4bit counter with 7seg LED  */
  /*******************************/
  module counter {
    /* 下位の部品タイプの定義 */
    circuit_type incre {
      input       in <4> ;
      output      out <4> ;
      instrin     up ;
      instr_arg   up (in) ;
      instruct    up       out = in + 0b1 ;
            /* 制御端子による動作の定義 */
    }
    circuit_type seven_seg {
      input   ind<4>;
      output  outd<7>;
      tmp     cnv<7>;
      instrin dis;
      instruct_arg dis(ind);
      instruct dis
        par {
          any {
            ind == 0x0 : cnv = 0b1111110;
            ind == 0x1 : cnv = 0b1100000;
            ind == 0x2 : cnv = 0b1011011;
            ind == 0x3 : cnv = 0b1110011;
            ind == 0x4 : cnv = 0b1100101;
            ind == 0x5 : cnv = 0b0110111;
            ind == 0x6 : cnv = 0b0111111;
            ind == 0x7 : cnv = 0b1100010;
            ind == 0x8 : cnv = 0b1111111;
            ind == 0x9 : cnv = 0b1110111;
            ind == 0xa : cnv = 0b1101111;
            ind == 0xb : cnv = 0b0111101;
            ind == 0xc : cnv = 0b0011001;
            ind == 0xd : cnv = 0b1111001;
            ind == 0xe : cnv = 0b0011111;
            ind == 0xf : cnv = 0b0001111;
          }
        outd = cnv;
        }
    }
    /* 構成要素の定義 */
    output  out <4> ;
                /* データ出力端子の定義 */
    instrin start ;
                /* 制御入力端子の定義 */
    reg     counter <4> ;
                /* レジスタの定義 */
    incre   inc ;
                /* 部品タイプ incre の使用定義 */
    seven_seg s_seg;
                /* 部品タイプ 7seg の使用定義 */
    output  disp <7> ;
                /* データ出力端子の定義 */
    /* 動作手順の存在と引数の定義 */
    stage_name  cnt {
        task    run (counter) ;
    }
    /* 動作手順の定義 */
    instruct start generate cnt.run(0b0000) ;
                /* 制御端子による動作の定義 */
    stage   cnt {
                /* ステージの動作手順の定義 */
      state_name  st1 ;
      state_name  st2 ;
      first_state st1 ;
                /* 初期状態の定義 */
      state st1 par {
                /* 状態の動作の定義 */
        counter := inc.up(counter).out ;
        out = counter ;
        disp = s_seg.dis(out).outd;
        goto st1 ;
      }
    }
  }
##############################################
```

1.6　4ビット加算器・減算器

以下に，4ビットの加算回路と加算演算を行なうためにスクリプトを示す．

```
++++++++++++++++++++++++++++++++++++
media% seconds
SFL> sflread 4adder.sfl
SFL> install four_adder /
SFL> add.sim ???? ????
             2進数によるデータ入力
++++++++++++++++++++++++++++++++++++
"add.sim"
====================================
set ci 0
hold ci
set in1 $1
hold in1
set in2 $2
hold in2
set ad 1
hold ad
pb res
====================================
####################################
module four_adder {
  circuit_type HA {
    input a, b;
    output s, c;
    instrin do;
    instruct_arg do(a, b);
    instruct do par{
      s = a @ b;
```

```
      c = ^(a & b);
    }
  }
  input in1<4>, in2<4>, ci;
  output res<4>, co;
  instrin ad;
  tmp p0, p1, p2, p3, gp;
  tmp g0, g1, g2, g3, gg;
  HA ad0, ad1, ad2, ad3;
  instruct ad par {
   p0 = ad0.do(in1<0>,in2<0>).s;
   g0 = ^ad0.c;
   p1 = ad1.do(in1<1>,in2<1>).s;
   g1 = ^ad1.c;
   p2 = ad2.do(in1<2>,in2<2>).s;
   g2 = ^ad2.c;
   p3 = ad3.do(in1<3>,in2<3>).s;
   g3 = ^ad3.c;
   res = (p3@(g2|(p2&g1)|(p2&p1&g0)|(p2&p1&p0&ci)))||
         (p2@(g1|(p1&g0)|(p1&p0&ci)))||
         (p1@(g0|(p0&ci)))||
         (p0@ci);
   gp=p3 & p2 & p1 & p0;
   gg=g3 |(p3 & g2)|(p3 & p2 & g1)|(p3 & p2 & p1 & g0);
   co=gg | (gp & ci);
  }
}
#####################################
```

1.6.1　符号付き演算

以下に４ビットで表現できる符号なし，符号付き数値を示す．

	符号なし数値	符号付き数値
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

コンピュータにおいて，負数は符号＋絶対値，１の補数，２の補数などの方法で表すことができ，このうち，どの表現方法を用いるかによって，減算器の構成は異なる．２の補数を用いて，上記のソースプログラムを改編して，減算器に作り換え，以下の計算を行なって，結果について考察せよ．なお，２の補数は，減数の各ビットを反転させ，最下位ビットに１を加える操作で作れるが，その操作は `ci` に１をセットすることで可能となる．

・符号なし演算 :

```
(1)  1 + 2    (2)  3 + 5    (3) 4 + 5    (4) 8 + 7
(5) 15 + 1    (6)  5 - 3    (7) 1 - 2    (8) 8 - 7
(9)  7 - 8   (10)  1 - 15
```

・符号つき演算 :

```
 (1)  2 + 3    (2)  3 + 5     (3)  4 + 5    (4) -8 + 8
 (5) -1 + 1    (6) -3 +(-5)   (7)  5 +(-3)  (8) -4 +(-5)
 (9)-8 +(-7)  (10)  5 - 3    (11)  1 - 2   (12) -8 - 7
(13) 7 -(-8)  (14)  1 -(-1)  (15) -3 - 5   (16) -4 - 5
```

[課題２]

上記加算器と，作成した減算器を一つにまとめ，加減算器を作成せよ．ただし，加算か減算を決めるスイッチを一つ設け，そのビットが０ならば，加算器，１ならば減算器となるようにせよ．(* ヒント Ex-OR の動作を考えてみよ)

1.6.2　キャリーフラグとオーバフローフラグ

コンピュータで２つの整数値を比較する場合などにキャリーフラグが用いられる．例えば，４ビットの演算において，ある変数が７以上かどうかを調べるために９を加え，桁上がりが生じたときにキャリーフラグが立つ．

また，その変数の値から８を引き，桁借りが生じた場合にもキャリーフラグが立つ．このように，汎用コンピュータにおいて，桁上がりと桁借りはキャリーフラグとして統一的に扱っている場合が多い．

上記の表から，４ビットで表現できる符号付き数値は−８から７までの数となる．演算の結果この範囲を越えた場合には，オーバフローが立つ．例えば，４＋５を実行すると結果は９であるが，２進数では `1001` となり，符号付きでは−７となる．この結果は符号付きの数値としては正しくないので，オーバフローフラグにより，ユーザに知らせる必要がある．

[課題３]

作成した加減算回路にキャリーフラグ，オーバフローフラグを検出する回路を付加せよ．また，その回路を用いて以下に示す演算結果について，それぞれどのフラグが立つか考察せよ．

・符号なし演算 :

```
(1)  1 + 2   (2) 3 + 5   (3) 4 + 5   (4) 8 + 7
(5) 15 + 1   (6) 5 - 3   (7) 1 - 2   (8) 8 - 7
(9)  7 - 8  (10) 1 - 15
```

・符号つき演算 :

```
 (1)  2 + 3    (2)  5 + 3     (3)  4 + 5    (4) -8 + 7
 (5) -1 + 1    (6) -3 +(-5)   (7)  5 +(-3)  (8)  3 +(-5)
 (9) -8 +(-7) (10)  5 - 3    (11)  1 - 2   (12) -8 - 7
(13)  7 -(-8) (14)  1 -(-1)  (15) -3 - 5   (16) -4 - 5
```

[課題４]

汎用のＣＰＵには他にもフラグが存在する．これらのフラグについて調べ，どのようにすれば実現できるかを考察せよ．