平成 20 年度前期 情報画像工学実験 II
実験 6. 論理回路設計

計算機システム工学教育研究分野 難波 一輝
1 号棟 409 室, 内線 3255
e-mail: namba@faculty.chiba-u.jp

計算機やディジタルシステムは論理回路から構成されている。情報画像工学実験 I 課題 4 では、回路図を用い論理回路の実装実験を行った。しかし、GPU (Graphics Processing Unit) などの巨大な回路を、回路図を用いて設計することは困難である。通常、巨大な回路は HDL(ハードウェア記述言語, Hardware Description Language) と呼ばれるプログラム言語によって、より抽象的、機能的に、記述される。本実験では、近年の回路設計技術に対する理解を深めるため、HDL の一つである Verilog-HDL を用いて論理回路設計を行う。また、シミュレータ上で設計した回路を動作させる。

本テキスト、次ページ以降では Verilog-HDL について説明する。本実験で使用するシミュレータについての説明、およびレポートについては以下の Web サイト上で説明する。

`http://www.icsd2.tj.chiba-u.jp/~namba/lecture/`

なお、本実験は計算機システム入門の教科書「入門計算機システム」[1] 、特に第 4 章、について、既に理解しているものとして構成されている。よく復習しておくこと。また、本実験は本テキストを充分予習しているものとして時間配分してある。よく予習しておくこと。

[1] 伊藤, 倉田, "—," 朝倉出版, 2000

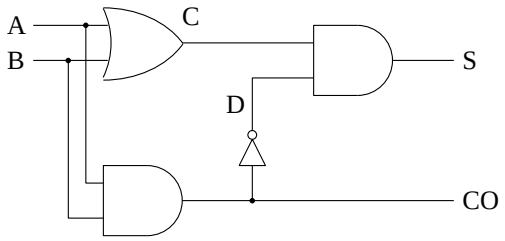

図 1: 半加算器 (回路図)

表 1: 半加算器の真理値表

A	B	CO	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

```
/* HALF_ADDER
     S : A + B
     CO : carry out */

module HALF_ADDER (A, B, S, CO);
    input A, B;
    output S, CO;

    wire C, D;

    assign C = A | B;    // 論理和
    assign CO = A & B;   // 論理積
    assign D = ~CO;      // 反転
    assign S = C & D;    // 論理積
endmodule
```

図 2: 半加算器 (Verilog-HDL 記述)

1 Verilog-HDL の基本

1.1 Verilog-HDL の生い立ち

Verilog-HDL は元々 Gateway Design Automation 社 (1989 年に Cadence 社に買収された) の開発した Verilog-XL というシミュレータに論理回路を入力するために開発された言語である。1990 年、Verilog-HDL の普及促進のために Open Verilog International[1]が組織され、1995 年に IEEE の標準ハードウェア記述言語[2]として採用された。その後見直しをされながら今日に至っている。

1.2 Verilog-HDL の記述例：半加算器

簡単な回路例として、図 1に半加算器 (half adder) と呼ばれる回路の回路図を示す。半加算器は 1 ビットの足し算を行う回路である。具体的に信号線 `A` と `B` の値を足して得られる値を信号線 `S` に返す。また、桁上げ (キャリー) を信号線 `CO` に返す。半加算器の真理値表を表 1 に示す。

[1]現 Accellera (`http://www.accellera.org/`)
[2]IEEE Std-1364

半加算器について、Verilog-HDL 記述を図 2に示す。Verilog-HDL 記述は回路図と異なり、全てテキストで回路情報を記述する。よって、C 言語のソースファイルのように、テキストエディタさえあれば、設計、修正を行うことができる。

1.3 コメント

Verilog-HDL では C++ 言語とまったく同様にコメントを書くことができる。すなわち `/*` から `*/` までの範囲はコメントである (例: 図 2, 最初の 3 行)。また、`//` から文末までの部分もコメントとして扱われる (例: 図 2, `assign` 文の後、日本語部)。

Verilog-HDL に限った話ではないが、後々の易読性を高めるため、コメントは積極的に書き残しておく方がよい。なお、図 2 中、日本語のコメントは書式例として示したものである。内容の悪いコメントなので真似すべきではない。一般にこのような情報性の非常に低いコメントをむやみに書き加えるべきではない。

1.4 識別子と予約語

C 言語における変数名、関数名のように、Verilog-HDL でも信号線名等、ユーザが自由に

つけることのできる名前が多数存在する。ユーザが自由に決められる名前のことを識別子と呼ぶ。識別子は以下のルールに基づいて決めることができる。

- 英数字、またはアンダースコア “_” のみからなる
- 先頭の一字は、英字またはアンダースコア
- 予約語でない

Verilog-HDL の識別子において大文字と小文字は区別される。すなわち、信号線 A と信号線 a は異なる信号線である。本書では予約語が全て小文字であることから、易読化のため、識別子は全て大文字で統一する。

1.5 モジュールと宣言部

1.5.1 モジュール

Verilog-HDL では、ひとまとまりの回路をモジュールと呼ぶ。モジュールは予約語 module で始まり、endmodule で終わる。module にはモジュール名が続き、全ての入出力信号名が列挙される。その後、endmodule までの間に回路内部についての記述がなされる。モジュールの書式を以下に示す。

```
module モジュール名
   (入出力信号名1, 入出力信号名2, ... );
   ＜回路内部の記述＞
endmodule
```

回路内部の記述は大きく宣言部と手続きブロックに分かれる。本文では宣言部についてのみ説明する。

1.5.2 input 文, output 文, wire 文

宣言部では、まず input 文, output 文 により、入出力信号の入力、出力の別、およびバス幅 (後述) を宣言する。

```
input [バス幅] 入力信号名;
output [バス幅] 出力信号名;
```

図 2に示したモジュール HALF_ADDER は、A, B という入力信号と、S, CO という出力信号を持つ。

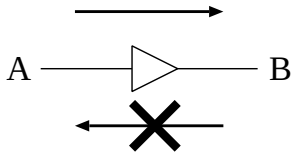

図 3: assign B = A; の示す回路

図 1に示した回路図において、外部との入出力信号の他に、回路内部で用いる二本の信号線 C, D が存在する。宣言部では、図 2 に示すように wire 文を用いて、内部の信号線を宣言しておく必要がある。入出力の宣言同様、必要がある場合にはバス幅 (後述) も宣言する。

```
wire [バス幅] 信号名;
```

1.5.3 assign 文 (継続代入文) と演算子

宣言部では assign 文により、回路の接続を記述できる。例えば

```
assign B = A;
```

と記述すれば、信号線 A を B に接続したことになる。ただし、assign 文で扱う信号は、等号の右辺から左辺への単方向の信号である。すなわち、この場合、図 3 のように A から B には信号は流れるが、B から A には信号は流れない。

assign 文の等号の右辺には、図 2 の様に、式を書くこともできる。具体的に、

```
assign C = A | B;
```

は、図 4 のように、信号 A と信号 B の論理和 (|) を取って得られる信号を信号 C に流すことを意味している。ここで、演算子として論理和を用いたが、他にも表 3 に示す演算子がある。また、本実験では用いないが、例えば算術演算子+, -, *, / 等、表 3 に示した他にも演算子が用意されている。

図 2 の例では、assign 文の等号の右辺の演算子の数は全て 1 個であったが、A & B | C のように、2 個以上の演算子を用いた表記も許される。このときの演算子の優先順位は

表 2: 予約語

always, and, assign, begin, buf, bufif0, bufif1, case, casex, casez, cmos, deassign, default, defparam, disable, edge, else, end, endcase, endmodule, endfunction, endprimitive, endspecify, endtable, endtask, event, for, force, forever, fork, function, highz0, highz1, if, ifnone, initial, inout, input, integer, join, large, macromodule, medium, module, nand, negedge, nmos, nor, not, notif0, notif1, or, output, parameter, pmos, posedge, primitive, pull0, pull1, pullup, pulldown, rcmos, real, realtime, reg, release, repeat, rnmos, rpmos, rtranif0, rtranif1, scalared, small, specify, specparam, strong0, strong1, supply0, supply1, table, task, time, tran, tranif0, tranif1, tri, tri0, triand, trior, trireg, vectored, wait, wand, weak0, weak1, while, wire, wor, xnor, xor

表 3: 演算子とその真理値表

~		&			|			^			? :	
反転		論理積			論理和			排他的論理和			条件演算子	
A	~A	A	B	A&B	A	B	A|B	A	B	A^B	A	A?B:C
0	1	0	0	0	0	0	0	0	0	0	0	C
1	0	0	1	0	0	1	1	0	1	1	1	B
		1	0	0	1	0	1	1	0	1		
		1	1	1	1	1	1	1	1	0		

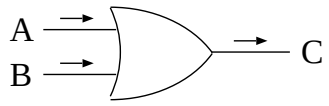

図 4: `assign C = A | B;` の示す回路

`~` > `&` = `^` > `|` > `?:`

である。優先順位が等しい演算は左から順に行われる。優先順位を変えたいときは括弧 `()` を用いるとよい。

Verilog-HDL の式の書き方は C 言語とほぼ同じである。とりあえず C 言語のつもりで式を書けば大概問題なく動く。

2 バス

1 本の信号線が表すことができる情報量は 1 ビットである。すなわち 0 か 1 のみである。2 ビット以上の情報を扱うためには 2 本以上の信号線を用いることになる。2 本以上の信号線をまとめた束をバスと呼ぶ。また、まとめた信号線の本数をバス幅と呼ぶ。

バスを使用するときには、`input`, `output`, `wire` などのキーワードと信号名の間にバス幅を記述する。例えば、

```
output [3:0] Y;
```

という記述は、 `Y[3]`, `Y[2]`, `Y[1]`, `Y[0]` なる 4 本の信号線を束ねたバスを出力 `Y` とすることを宣言している。なお、バス幅の指定は文全体に対して有効である。例えば、

```
wire [3:0] X, Y;
```

という記述は、`X[3]`, `X[2]`, `X[1]`, `X[0]` なる信

号線からなるバスと、Y[3], Y[2], Y[1], Y[0] なる信号線からなるバスの宣言を意味する。異なる幅を有する複数のバスを宣言したいときは、複数の宣言文に分けて記述する他ない。

既に述べたように、モジュールの宣言において入出力信号線名を列挙する必要がある。バスを入出力信号として用いるときは、以下の例のように、バス名のみを列挙すれば良く、バス幅を書いてはいけない。

```
module ADDER (A, B, S, CO);
  input [3:0] X, Y;
  output [3:0] S;
  output CO;
  ...
```

assign 文 において、信号線の代わりに同一幅のバスを記述することができる。この場合、各ビットについてそれぞれ assign 文 を記述した場合と等価になる。すなわち、例えば幅が 4 であるバス A, B, Y について

```
assign Y = A & B;
```

は

```
assign Y[3] = A[3] & B[3];
assign Y[2] = A[2] & B[2];
assign Y[1] = A[1] & B[1];
assign Y[0] = A[0] & B[0];
```

と等価である。

バスを構成する全ての信号線に同一の演算子を適用し一本の信号線を得たいとき、まず演算子を書き、続けてバス名を書けばよい。すなわち、例えば幅が 4 であるバス A、および信号線 Y について

```
assign Y = A[3] | A[2] | A[1] | A[0];
```

と

```
assign Y = | A;
```

は等価である。

複数の信号線、およびバスをカンマ (,) で区切りながら列挙し、中括弧 {} で括ることにより、これらの信号線、バスを束ねた新たなバスを作ることもできる。

例えば、幅が 3 であるバス A と、信号線 B, C, D について

```
assign A = {B, C, D};
```

は

```
assign A[2] = B;
assign A[1] = C;
assign A[0] = D;
```

と等価である。

表 4: 全加算器の真理値表

A	B	CIN	CO	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

3 階層化記述

図 6 に全加算器 (full adder) とよばれる回路の回路図を示す。全加算器は半加算器同様、信号線 A, B の和を S、桁上がりを CO に返す回路である。ただし、下位桁からの桁上げ CIN を加えることができる点で半加算器と異なる。A, B, CIN の 3 値の和を S に、桁上げを CO に返す回路と理解してもよい。全加算器の真理値表を表 4 に示す。

A+B+CIN を計算するためには、まず半加算器で A+B を計算し、得られた和 U0_S と CIN の和を半加算器で計算すればよい。すなわち、全加算器について、真理値表を見ながら、一から回路構成を考える必要はなく、図 6 のように半加算器を 2 個用いれば容易に構成できる。

Verilog-HDL では、モジュールを構成するとき、既に構成した別のモジュールを部品として用いることができる。図 5に半加算器を用いて構成した全加算器の Verilog-HDL 記述を示す。

別のモジュールを部品として用いるときの書式は以下の通りである。

```
module FULL_ADDER (A, B, CIN, S, CO);
    input A, B, CIN;
    output S, CO;

    wire U1_CO, U1_S, U2_CO;

    HALF_ADDER U1 (.A(A), .B(B), .S(U1_S), .CO(U1_CO));
    HALF_ADDER U2 (.A(U1_S), .B(CIN), .S(S), .CO(U2_CO));

    assign CO = U1_CO | U2_CO;
endmodule
```

図 5: Verilog-HDL 記述

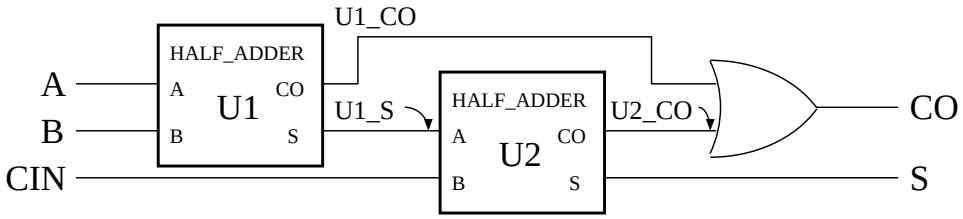

図 6: 回路図

```
モジュール名 インスタンス名
        (. ポート名 (信号名), ...);
```

モジュール名は部品として用いたいモジュールの名前である。インスタンス名は、各インスタンス (部品として用いられるモジュール) を識別するための名前であり、モジュール内でユニークな識別子を付ける。各インスタンスの入出力と信号線の接続は “. ポート名 (信号名)” という形式で行う。例えば、図 5 のインスタンス U1 における .A(U0_S) は、HALF_ADDER の入力 A に信号線 U0_S を接続することを意味する。