Artgraphics

SystemVerilog による論理合成

ハードウェア記述言語による回路設計の基礎

Document Identification Number: ARTG-TD-001-2024
Document Revision: 2.8, 2025.06.30
アートグラフィックス
篠塚一也

SystemVerilog による論理合成
ハードウェア記述言語による回路設計の基礎

© 2025 アートグラフィックス
〒124-0012 東京都葛飾区立石 8-14-1
www.artgraphics.co.jp

Logic Synthesis with SystemVerilog

© 2025 Artgraphics. All rights reserved.
8-14-1, Tateishi, Katsushika-ku, Tokyo, 124-0012 Japan
www.artgraphics.co.jp

注意事項

- 本解説書により得られた知識・情報の使用から生じるいかなる損害についても、弊社および本書の著者は責任を負わないものとします。
- 弊社の技術資料の内容の一部、あるいは全部を無断で複写、複製、転載する事は、禁じます。
- 弊社の技術資料の譲渡、転売、模倣、又は、改造等の行為を禁止致します。

はじめに

今日では、ハードウェア記述言語（HDL）はハードウェア設計・検証に欠かせない手段となっています。設計者はHDLを使用してデザインをすすめます。そして、HDLにより記述された RTL デザインは検証されて、論理合成によりゲートレベルのネットリストに変換されます。本書は、ハードウェア記述言語で書かれた内容がハードウェアに変換される仕組みを分かり易い理論で解説し、ハードウェア記述言語を正しく理解するための知識と技術を養います。特に、ブール式を用いて最適化するための計算手順を紹介しデザインの回路構成を予測する手段を提供します。

先ずは、ハードウェア、ハードウェア記述言語、論理合成との関係を簡単な事例を用いて考察する事から始めます。`a==b` と記述すると XNOR が生成されることは周知の事実であると思います。直ぐにはピンと来ないかも知れませんが、以下のように書いても、やはり XNOR が合成されます。

```
{a,b} == 2'b00 || {a,b} == 2'b11
```

上記の記述が XNOR を表現している事実は以下に示す XNOR の真理値表により確認できます。この記述は、単に真理値法を表現し直しているだけなので HDL の特性を活用していないと言えます。しかも、このような記述をしてしまう事もあるので、注意が必要です。

(a) XNOR の真理値表

a	b	out
0	0	1
0	1	0
1	0	0
1	1	1

同様に、以下の記述により XOR が合成されます。やはり、この種の冗長な記述法もしばしば見られるので、注意が必要です。

```
{a,b} == 2'b01 || {a,b} == 2'b10
```

XNOR と同様に、以下に示す XOR の真理値表により、上記の記述が XOR を表現している事実を確認できます。

(b) XOR の真理値表

a	b	out
0	0	0
0	1	1
1	0	1
1	1	0

上記で指摘した冗長な記述現象は SystemVerilog 特有な問題ではなく Verilog HDL でも同様に起こる問題です。つまり、HDL 特有の記述問題といえます。以上をまとめると、以下のような事実が判明してきます。

冗長な HDL 記述は真理値表を単純に描写している可能性がある。

そもそも、HDL はハードウェアの実装を意識せずに高度な概念の下で機能的な設計を進めるために存在します。したがって、わざわざ真理値表を HDL で記述するのは HDL 本来の目的から逸脱していると言えます。したがって、このような不注意を防ぐための対策を身に付けておかなければなりません。そのためには、基本回路の持つ機能を正しく理解しておく必要があります。例えば、上記に現れている`{a,b} == 2'b01` は、`a < b` を意味します。この事実

Copyright © 2025 Artgraphics. All rights reserved.

は、以下に示す真理値表により確認できます。

(c) a < b の真理値表

a	b	out
0	0	0
0	1	1
1	0	0
1	1	0

同様に、`{a,b} == 2'b10` は `a > b` を意味します。この事実は、以下に示す真理値表により確認できます。

(d) a > b の真理値表

a	b	out
0	0	0
0	1	0
1	0	1
1	1	0

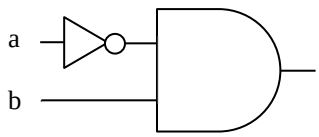
したがって、先ほどの XOR 記述を以下のようにも記述できる事が分かります。

```
a < b || a > b
```

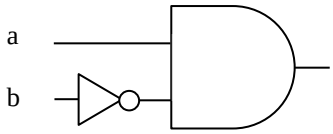
このように表現すると、寧ろ、確実に以下のような記述に落ち着くと思えます。

```
a != b
```

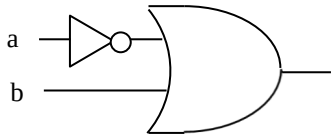
以上の事例から、冗長な表現は基本演算が表現する回路機能を十分に理解していれば避けられると言えます。因みに、特に重要な基本回路としては以下のような回路があります。AND と OR にインバータを付加するだけで高度な機能を持つ基本回路に変化します。



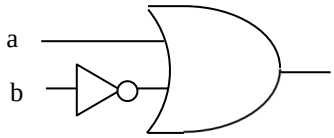
(e) a < b



(f) a > b



(g) a <= b



(h) a >= b

表現式`{a,b} == 2'b10` を見ていても新しい発想は何も浮かんできませんが、この式を `a > b` と書き換えれば、多くの活用法が浮かんできます。つまり、物質的な表現法を避け、機能的な表現法に徹すると無限の可能性に結び付く事を示唆しています。

さて、一般に、論理合成は万能ではありません。意図に即した最適な回路構成を得るためには、合成し易い記述が与えられなければなりません。例えば、jk―フリップフロップを以下に示す(i)のように記述すると、論理合成には難しい課題を与えるため最適な回路構成を得られるか否かは論理合成ツールの性能に依存します。しかし、(j)のように記述すれば、殆どの場合に最適な回路に合成されます。

Copyright © 2025 Artgraphics. All rights reserved.

(i) 論理合成を考慮しない記述	(j) 論理合成を考慮した記述

`always @(posedge clk) begin`
`  if( j == 1'b0 && k == 1'b0 )`
`  q <= q;`
`  else if( j == 1'b0 && k == 1'b1 )`
`  q <= 1'b0;`
`  else if( j == 1'b1 && k == 1'b0 )`
`  q <= 1'b1;`
`  else if( j == 1'b1 && k == 1'b1 )`
`  q <= ~q;`
`end`
	`always @(posedge clk) begin`
	`  if( {j,k} == 2'b00 )`
	`  q <= q;`
	`  else if( {j,k} == 2'b01 )`
	`  q <= 1'b0;`
	`  else if( {j,k} == 2'b10 )`
	`  q <= 1'b1;`
	`  else if( {j,k} == 2'b11 )`
	`  q <= ~q;`
	`end`

では、どのような回路に合成されるか予測してみます。jk―フリップフロップの仕様から下表(1)を得ます。表(2)は、現在の状態 `q(t)` が 0 である場合の次の状態 `q(t+1)` を示し、表(3)は現在の状態 `q(t)` が 1 である場合の `q(t+1)` を示しています。

j	k	q(t+1)
0	0	q(t)
0	1	0
1	0	1
1	1	~q(t)

(1) jk―フリップフロップの真理値表

j	k	q(t+1)
0	0	0
0	1	0
1	0	1
1	1	1

(2) q(t) == 0 の時

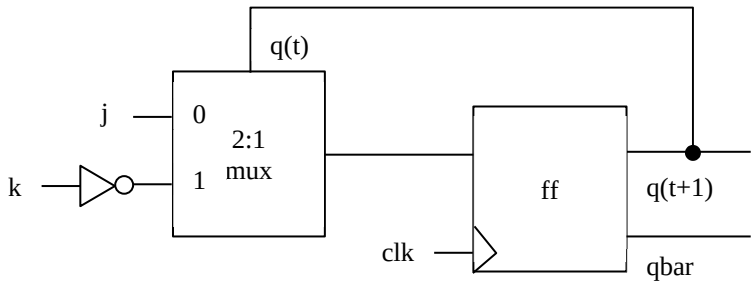
j	k	q(t+1)
0	0	1
0	1	0
1	0	1
1	1	0

(3) q(t) == 1 の時

上表から次の事がわかります（詳しい解説は 1.5.1 項にあります）。

- 表(2)から、`q(t)==0` であれば、`q(t+1)==j` である。
- 表(3)から、`q(t)==1` であれば、`q(t+1)==~k` である。

したがって、この回路は 2:1 マルチプレクサにより構築できる事が分かります。以上の解析により、下図に示す回路構成が得られます。



本書は、基本的な構文を利用して、RTL 記述がどのような回路に合成されるかを習得する技術を養う目的を持っています。そのために、RTL 記述に必要な SystemVerilog の基礎知識、論理合成が仮定しているルール、組み合わせ回路の記述法、シーケンシャル回路の記述法、ラッチの記述法など幅広い観点から解説を進めています。

第 1 章では、ハードウェア記述言語、回路設計、論理合成、論理の最適化を概説していますが、この章は本書の中で最も難しい内容となっています。他の章の解説は、第 1 章の解説を理論的に裏付けする目的を持っています。

第 2 章では、SystemVerilog によるモデリングで必要な SystemVerilog に関する知識を復習します。SystemVerilog 機能の効果的記述法も含まれているので一通り目を通すことをすすめます。例えば、従来の思考法では冗長になりがちであった表現も新しい発想の下では簡潔明瞭に表

Copyright © 2025 Artgraphics. All rights reserved.

現できる手法が紹介されています。

第 3 章では、真理値表とブール代数に関しての知識を総括しています。組み合わせ回路を設計する場合、真理値表からブール式を導く必要性も少なからず出て来ると想定して、この章にはブール代数の基本的な知識をまとめてあります。特に、Shannon の展開定理に関係するロジック設計の代表的な話題がまとめられています。

第 4 章では、RTL の動作記述をするための構文 `always` の使用法をセンシティビティリスト、ブロッキング代入文、ノンブロッキング代入文と関連させて詳しく解説しています。`always` の意義を正しく理解するための重要な章です。

第 5 章では、モデリングで必要とされるエンコーディングが解説されています。Gray コード、Johnson コード、one-hot コード等に関する基礎知識を再確認するために役立つ知識が解説されています。

第 6 章では、データ表現の基本となる整数表現を詳しくまとめてあります。特に、重要な概念である整数演算オーバーフロー検出を解説してあります。解説した知識を明確にするために、この章には演算オーバーフロー検出機能を使用した加減算器のデザインも含まれています。また、加減算器の応用例としてインクリメントとデクリメントの二つの機能を備えた回路も紹介します。

第 7 章は、RTL 記述の基礎を構成する演算子と式の解説を含んでいます。その他、`if` 文や `case` 文の合成技術の基礎知識が詳しく解説されています。

第 8 章は、ラッチに関する話題です。レベルセンシティブなラッチと RS ラッチに関する注意事項が詳しく解説されています。

第 9 章は、シーケンシャル回路のモデリングと論理合成を主題にしています。シーケンシャル回路の検証法の簡単な解説も含まれています。

第 10 章は、FSM の解説が主題です。FSM のモデル定義から始まり、FSM の例を紹介し、ハードウェア記述言語による FSM の実装法を詳しく解説しています。FSM の厳密な定義と具体例により Moore FSM と Mealy FSM の相違が明確になります。また、デザインした FSM をテストする際の相違に対しての解説も含んでいます。Mealy FSM は、クロックと非同期に動作するので、Moore FSM と同じ方法でテストをする事ができません。両者のモデリングによる結果を比較し易くするために、テストベンチも両 FSM の差異を意識して作りました。この章のモデリングとテストベンチ記述法を理解する事は、FSM の完全理解に繋がります。

本書は、論理合成のルールと RTL 記述を回路に変換する技術を様々なデザイン例を通して詳しく解説しています。本書を精読することにより、SystemVerilog による適切な RTL 表現を準備する習慣が自然と身に付きます。

アートグラフィックス
篠塚一也

Copyright © 2025 Artgraphics. All rights reserved.

変更履歴

日付	Revision	変更点
2024.11.22	1.0	初版。
2025.01.17	1.1	全体的な校正を実施。
2025.03.14	2.0	全体的に内容を改訂。
2025.03.31	2.1	練習問題を追加。
2025.04.07	2.2	全体的な校正を実施。
2025.04.30	2.3	第 6 章を改訂。
2025.05.31	2.4	全体的な改訂を実施。
2025.06.05	2.5	第 1 章の節を並び替えロジック関数を最初に解説した。
2025.06.08	2.6	ブール代数の定義を第 1 章に移動。
2025.06.25	2.7	第 3 章にブール式の最適化手順を追加。
2025.06.30	2.8	3.1 節を改訂。

Copyright © 2025 Artgraphics. All rights reserved.

目次

1 概要 1
1.1 ブール代数 1
1.1.1 ブール代数の定義 1
1.1.2 ブール代数の重要な性質 1
1.1.3 Shannon の展開定理（Boole の展開定理） 2
1.2 2 変数ロジック関数 3
1.3 SystemVerilog による動作記述 4
1.4 2 変数ロジック関数とマルチプレクサ 7
1.5 ロジック関数による論理合成 8
1.6 回路予測 13
1.6.1 ヒューリスティックな予測法 13
1.6.2 RTL 記述から予測する方法 14
1.6.3 正攻法 15
1.7 合成回路の予測技術 16
1.8 最適化の手段 16
1.9 本書の目的と構成 17
1.10 例題に関して 19
1.11 本書の記法 19
1.12 練習問題 21
2 設計に必要な SystemVerilog の基礎知識 24
2.1 基礎知識 24
2.1.1 論理値 24
2.1.2 4-state 型 25
2.1.3 2-state 型 27
2.1.4 enum データタイプ 29
2.1.5 string データタイプ 30
2.1.6 可変長リテラル（'0、'1、'x、'z、'X、'Z） 31
2.1.7 string リテラル 32
2.1.8 値としての$ 33
2.1.9 ディレー 33
2.1.10 packed アレイと unpacked アレイ 33
2.1.10.1 packed アレイ 34
2.1.10.2 unpacked アレイ 35
2.1.11 ビットセレクトとパートセレクト 35
2.1.12 プロシージャ 36
2.1.12.1 センシティビティリスト 36
2.1.12.2 センシティビティリストの指定 36
2.1.12.3 always プロシージャの種類 37
2.1.13 モジュール 38
2.1.13.1 モジュールの定義 38
2.1.13.2 ポートの宣言 38
2.1.13.3 ポートの方向に関するルール 39
2.1.13.4 ポートの種類 39
2.1.13.5 モジュールインスタンス 40
2.2 モデリングに役立つ基礎知識 40
2.2.1 オペレータとオペランド 41
2.2.2 {} オペレータ 42
2.2.3 inside オペレータ 44
2.2.4 ビットセレクト 45
2.2.5 プライオリティとパラレル 47
2.2.6 ブロッキング代入文とノンブロッキング代入文 48

Copyright © 2025 Artgraphics. All rights reserved.

2.2.7 パラメータとパッケージによる汎用的な記述 ... 49
2.2.7.1 パラメータによる汎用的記述 ... 49
2.2.7.2 パッケージによる汎用的記述 ... 50
2.3 ループ処理の記述 ... 51
2.4 練習問題 ... 53

3 ブール代数と論理合成 ... 56

3.1 ブール代数の応用 ... 56
3.2 Shannon の展開定理と if-then-else ... 57
3.3 Shannon の展開定理の応用 ... 58
3.3.1 マルチプレクサによる階層設計 ... 58
3.3.2 Shannon の展開定理と XOR ... 58
3.3.3 Shannon の展開定理とマルチプレクサ ... 59
3.4 ハーフアダーとフルアダー ... 60
3.4.1 ハーフアダーと真理値表 ... 60
3.4.2 フルアダーと真理値表 ... 61
3.4.3 ハーフアダーとフルアダーの計算アルゴリズムのまとめ ... 63
3.4.4 ハーフアダーとフルアダーの活用法 ... 63
3.5 Shannon の展開定理とブール式の最適化 ... 65
3.5.1 3 変数のロジック関数の最適化手順 ... 65
3.5.2 4 変数のロジック関数の最適化手順 ... 67
3.6 練習問題 ... 69

4 論理合成と always ... 72

4.1 センシティビティリスト ... 72
4.2 組み合わせ回路 ... 72
4.2.1 センシティビティリスト ... 73
4.2.2 ブロッキング代入文 ... 73
4.2.3 シングルビットとマルチビット ... 74
4.2.4 always_comb ... 75
4.3 シーケンシャル回路 ... 76
4.3.1 センシティビティリスト ... 77
4.3.2 ノンブロッキング代入文 ... 80
4.3.3 フリップフロップ数 ... 80
4.4 練習問題 ... 82

5 エンコーディング ... 85

5.1 Gray コード ... 85
5.2 Johnson コード ... 87
5.3 one-hot コード ... 88
5.4 練習問題 ... 89

6 整数表現と演算 ... 91

6.1 整数表現 ... 91
6.2 signed と unsigned ... 92
6.3 符号の反転処理 ... 94
6.4 加減算の精度 ... 97
6.5 演算とオーバーフロー* ... 98
6.6 練習問題 ... 104

7 組み合わせ回路の論理合成 ... 105

7.1 演算子 ... 105
7.1.1 ビット演算 ... 105
7.1.2 関係演算子 ... 108

Copyright © 2025 Artgraphics. All rights reserved.

7.1.3 数値演算 ... 110
7.2 if/case 文の合成 ... 110
7.3 デコーダー ... 113
7.4 記述のプライオリティ ... 116
7.5 3 ステートバス ... 118
7.6 記述の改良 ... 119
7.7 練習問題 ... 120

8 ラッチ* ... 122

8.1 ラッチの記述法 ... 122
8.2 ラッチを回避する方法 ... 123
8.3 RS ラッチ ... 123
8.4 RS ラッチと信号名 ... 125
8.5 RS ラッチとマルチビット ... 125
8.6 練習問題 ... 128

9 シーケンシャル回路の論理合成 ... 130

9.1 シーケンシャル回路のモデリング ... 130
9.2 シーケンシャル回路の検証法 ... 130
9.3 非同期信号を持つシーケンシャル回路* ... 132
9.4 非同期信号とマルチビット* ... 133
9.5 データシフタ ... 135
9.6 練習問題 ... 137

10 FSM* ... 139

10.1 FSM と状態遷移図の定義 ... 139
10.2 FSM の例 ... 140
10.2.1 カウンター ... 140
10.2.2 パリティチェッカー ... 140
10.2.2.1 パリティチェッカーの概要 ... 140
10.2.2.2 Moore FSM による次の状態と出力計算 ... 141
10.2.2.3 Mealy FSM による次の状態と出力計算 ... 141
10.3 FSM の種類 ... 142
10.4 Moore FSM ... 143
10.4.1 Moore FSM の構成 ... 143
10.4.2 Moore FSM のモデリング構造 ... 144
10.5 Mealy FSM ... 144
10.5.1 Mealy FSM の構成 ... 144
10.5.2 Mealy FSM のモデリング構造 ... 145
10.6 FSM の適用例 ... 145
10.6.1 パターン認識の問題 ... 145
10.6.2 Moore FSM によるモデリング ... 146
10.6.2.1 Moore FSM によるパターン認識問題の状態遷移図 ... 146
10.6.2.2 Moore FSM 記述例 ... 146
10.6.2.3 テストベンチ ... 147
10.6.2.4 実行結果 ... 148
10.6.3 Mealy FSM によるモデリング ... 148
10.6.3.1 Mealy FSM によるパターン認識問題の状態遷移図 ... 148
10.6.3.2 Mealy FSM 記述例 ... 149
10.6.3.3 テストベンチ ... 150
10.6.3.4 実行結果 ... 150
10.7 練習問題 ... 151

11 練習問題の解答 ... 153

11.1 第 1 章の練習問題の解答 ... 153

Copyright © 2025 Artgraphics. All rights reserved.

11.2 第 2 章の練習問題の解答 157
11.3 第 3 章の練習問題の解答 160
11.4 第 4 章の練習問題の解答 164
11.5 第 5 章の練習問題の解答 165
11.6 第 6 章の練習問題の解答 168
11.7 第 7 章の練習問題の解答 170
11.8 第 8 章の練習問題の解答 176
11.9 第 9 章の練習問題の解答 177
11.10 第 10 章の練習問題の解答 179

12 参考文献 184

本書で使用する略語一覧

略語	定義
ALU	Arithmetic Logic Unit
BCD	Binary Coded Decimal
DUT	Design Under Test、または、Device Under Test
EDA	Electronic Design Automation
FSM	Finite State Machine
HDL	Hardware Description Language
LHS	Left Hand Side
LRM	Language Reference Manual、すなわち、IEEE Std 1800-2023
LSB	Least Significant Bit
MSB	Most Significant Bit
NBA	Nonblokcking Assignment
POS	Products of Sum
RHS	Right Hand Side
RTL	Register Transfer Level
SOP	Sum of Products

Copyright © 2025 Artgraphics. All rights reserved.