計算機科学実験及び演習 2(CAD ツール上での論理設計)

第 3 回

締め切り：1 月 28 日

提出日：1 月 26 日

学籍番号: 1029300562
氏名: 新山公太（にいやまこうた）

1　ALU の設計

今回の課題は加減算及び基本的な論理演算を備えた ALU を作ることであった。また、発展課題として比較、移動演算を備えることが要求された。今回はこれらすべての機能を備えた ALU を作成したので、その仕様を満たしているか、また性能はどの程度であるかについてをレポートにまとめる。なお、今回作った ALU は全加算器の繰り上げを順次次の加算器へと受け渡していく方式 (桁上げ伝播方式) をとった。高速化するためには、桁上げ先見器を用いることが考えられるがこれは回路規模とのトレードオフとなる。先見器を用いた構成は実際に設計をするところまではできなかったので、実際にシミュレーションした値で性能を比較することはできないので、文献をもとにどのくらいの性能差があるかを論述するにとどめる。この際に参考にした文献については、巻末にまとめて示す。

1.1　設計した ALU の仕様

配布されたプリントに記されている仕様の再掲となる部分も多いが、すべての入出力について説明していきたいと思う。まず dataA,dataB はそれぞれ 16 ビットの入力であり、このデータをもとに演算が行われる。次に opcode では操作コードが指定され、配布資料にある通りに演算が指定される。なお、opcode は 4 ビットの入力信号であるが、与えられた操作コードは 7 個であり、対応してない入力が存在する。そのようなものについては動作を保証していない。対応してない入力に関してはエラーを返す仕様にすることもできたが、回路規模が多くなる恐れがあったので、今回はその機能を採用することはしなかった。

1.2　シミュレーション結果

1.3　加算

図 1 を見ればわかるようにちゃんと 2 進数の足し算が実現されている。cond についてもすべてしっかりと判定できている。また、線が赤色の所は冗談からの桁上げが伝播していないために計算が終わっていないところがあることを示している。

1.4　減算

減算は dataB の補数を取って加算を行っている。図を見れば正しい出力が得られていることがわかる。なお、cond のうち、桁上げを表すフラグが立っているが、上述の通り補数を取って加算をしているためである。

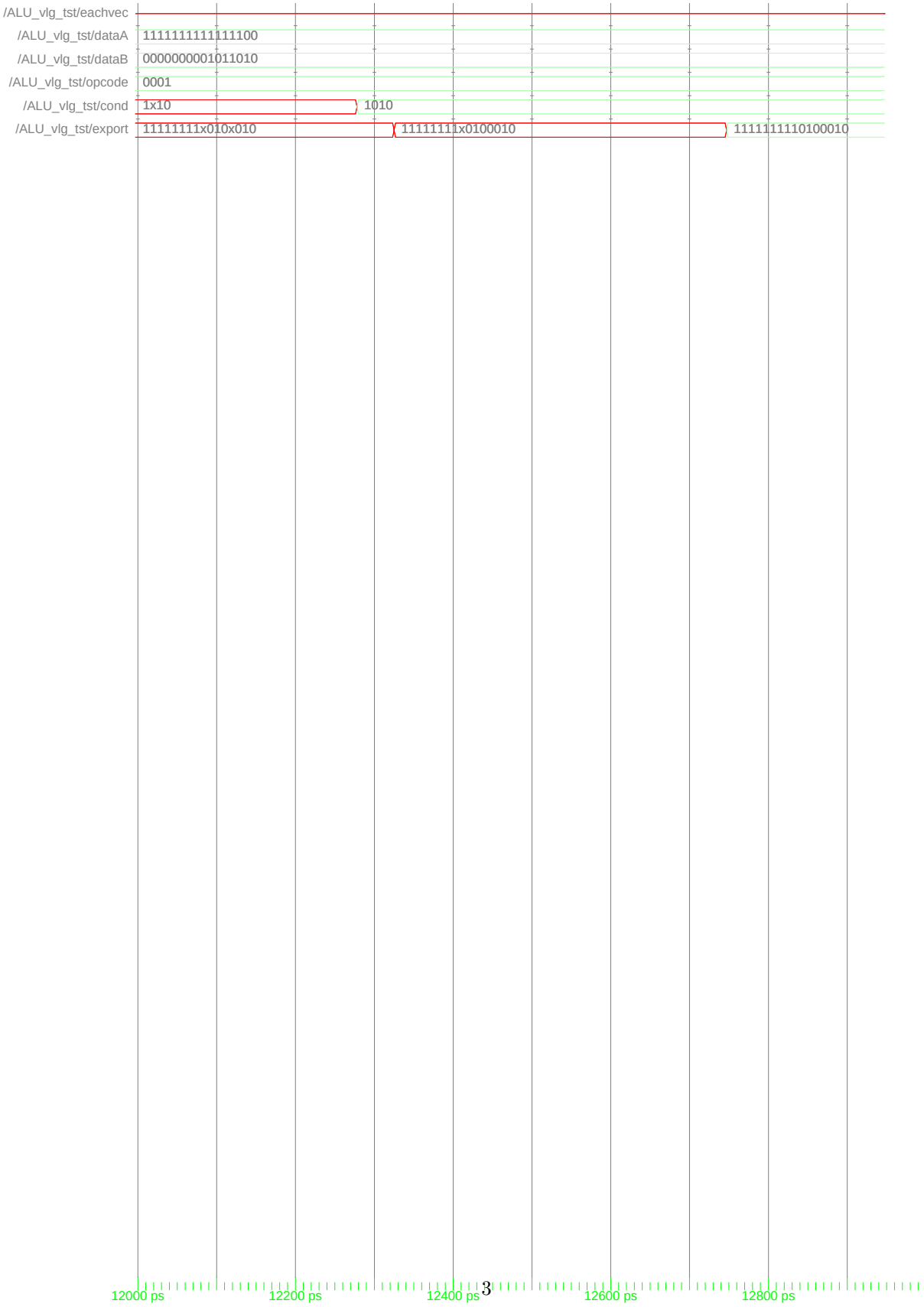
1.5　論理積

出力に関しては図を見れば正しい答えが得られていることがわかる。気になるのは、オーバーフローのフラグがたっていることであるが、この ALU の設計では操作コードによらず任意の演算を実行しておいて、後で操作コードによって採用する出力をマルチプレクサによって選択しているが、オーバーフローの判定は算術演算の結果を流用して行っており、論理演算が選択されたときについてはオーバーフローすることがないのでドントケアにしていたためである。マルチプレクサを使って算術演算が選ばれたときと論理演算が選ばれたときで分岐させてもよいと思ったが、仕様を確認した結果ドントケアのままでも問題ないと判断した。今回は回路

図 1　加算

Entity:ALU_vlg_tst Architecture: Date: Fri Jan 24 03:09:29 +0900 2020 Row: 1 Page: 1

図 2　減算


/ALU_vlg_tst/eachvec
/ALU_vlg_tst/dataA
1111111111111100
/ALU_vlg_tst/dataB
0000000001011010
/ALU_vlg_tst/opcode
0001
/ALU_vlg_tst/cond
1x10
1010
/ALU_vlg_tst/export
11111111x010x010
11111111x0100010
1111111110100010
12000 ps
12200 ps
12400 ps
12600 ps
12800 ps

規模がそんなに大きくないので問題にはならないが、意図してない入力による出力はドントケアとすることで回路規模を小さくすることができる。

1.6　論理和

特に言及することはなく図から正しい結果が返されていることがわかる。

1.7　排他的論理和

図から各ビットについてちゃんと排他的論理和がとられていることがわかる。

1.8　比較演算

等しい入力を与えると Z に 1 がたっていることがわかる。

1.9　移動演算

dataA の中身がそのまま出力されていることがわかる。

1.10　オーバーフロー

加算についてオーバーフローする例についてシミュレーションした結果、cond のオーバーフローフラグにちゃんと 1 がたっていることが確認できる。

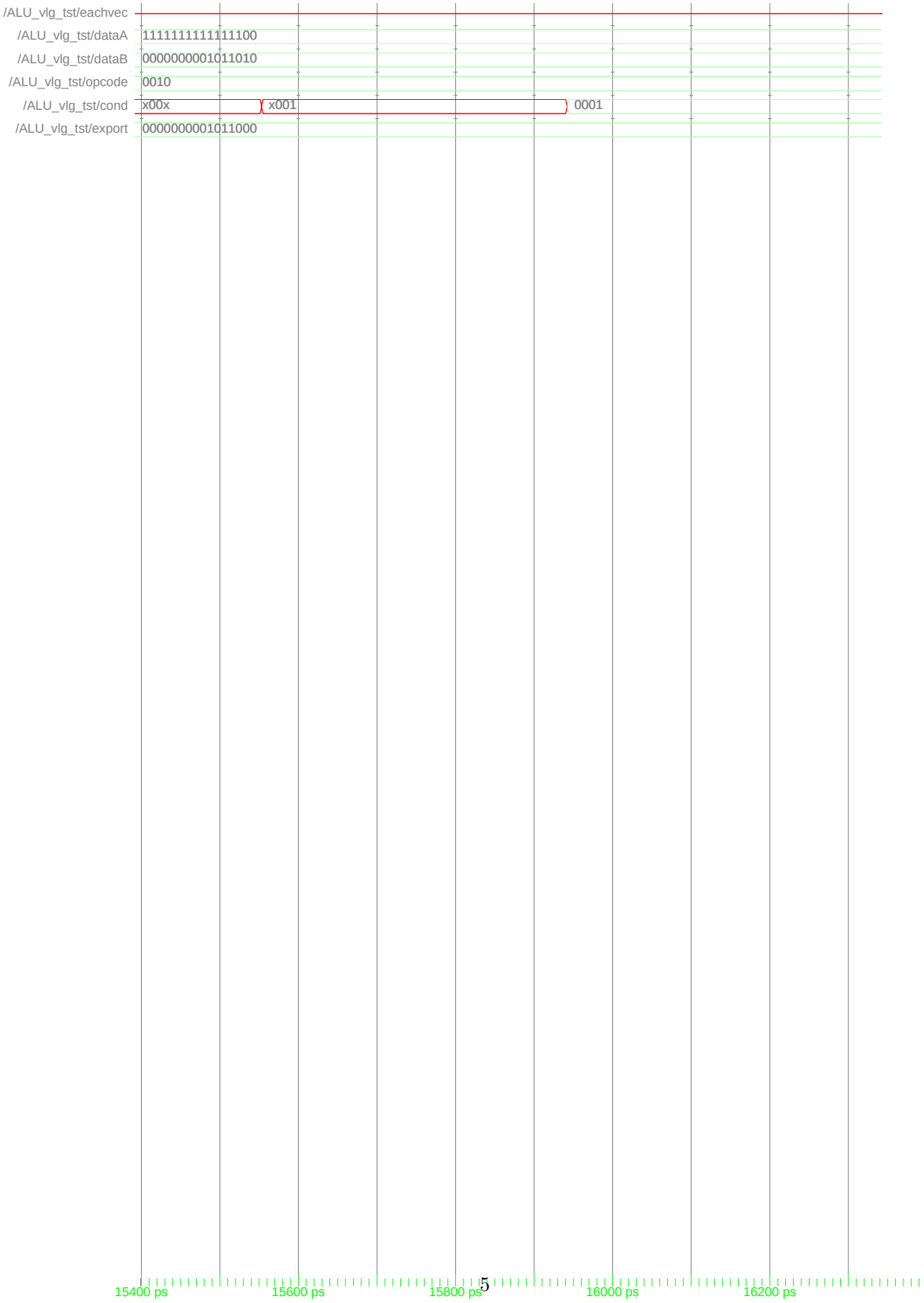
1.11　改良案 (先見回路を用いた例)

上記の ALU では桁上げ伝搬加算器を用いて ALU を構成したため、論理素子が多段接続される数が多くなり遅延が大きくなる。これを解消するためには桁上げ先見加算器を用いた構成を用いることが考えられる。これは桁上げ信号が伝播する論理段数を減らすことで高速化を図るものである。回路規模を気にしなければ当然 16 ビット先見加算器を用いればよいのだが、桁数が大きくなるとキャリーを事前に求めるための論理回路が大きくなるので現実的には 4 ビット先見加算器程度にするのが良いだろう。回路規模の点では桁上げ伝播加算器の方が性能が良いといえるが、計算時間に関しては桁上げ先見加算器のほうが性能が良いといえる。`https://ocw.kyoto-u.ac.jp/ja/09-faculty-of-engineering-jp/computer-architecture-basics/pdf/slide6.pdf` によると、桁数を n として、桁上げ伝搬加算器においてはオーダーは O(n), 桁上げ先見加算器については O(log n) である。

2　ALU を用いた同期式順序回路の設計 (1001 の検出)

この課題では上で作った ALU を用いて順序回路を作成した。使った機能は必須機能に限って簡単な単語を受理する順序機械を構成した。クロックに合わせて入力を入れてやらないと入力が何回も読み込まれるようになっているため、テストベンチファイルのほうで上手く調整することで対応した。

図 3　論理積

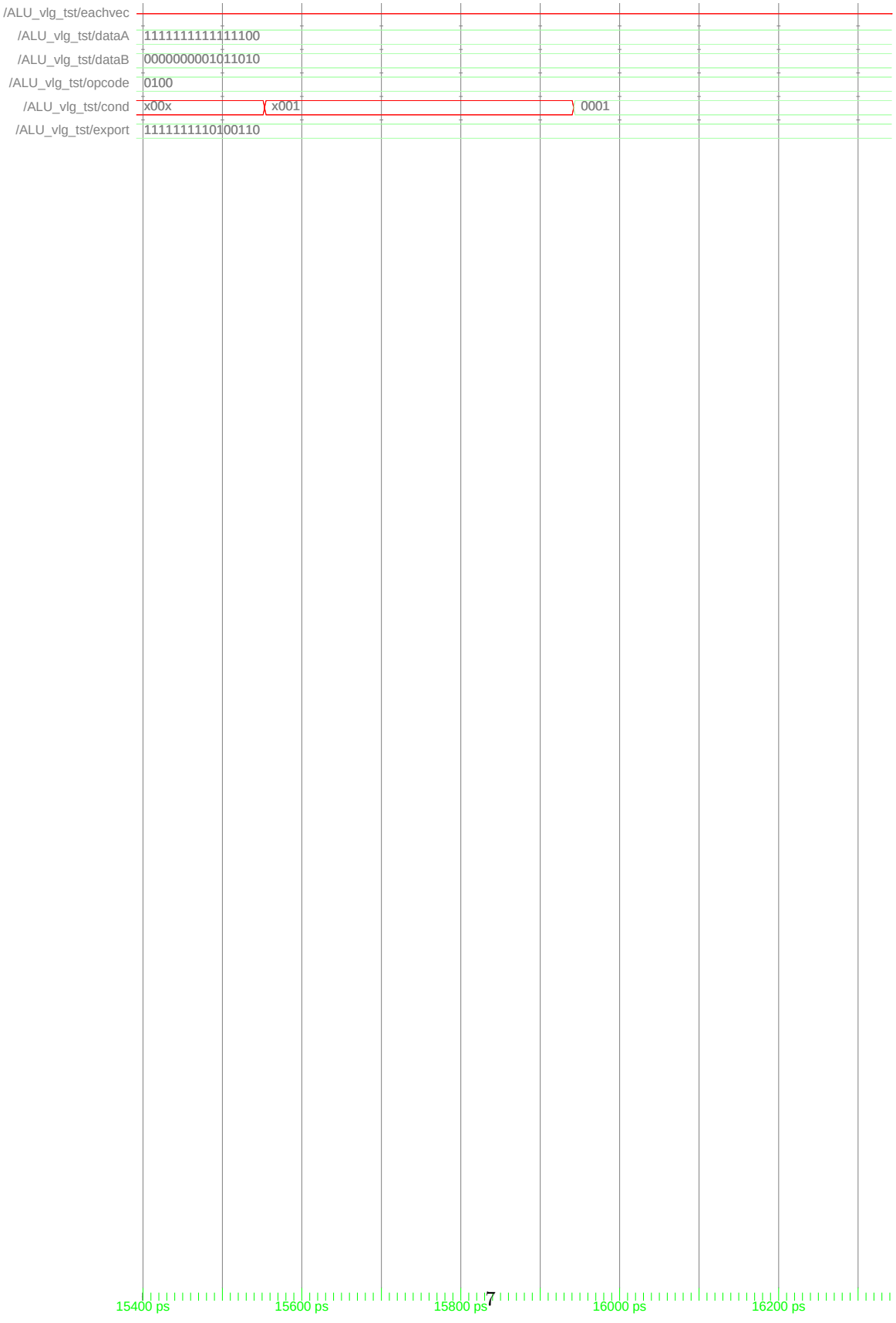


Entity:ALU_vlg_tst Architecture: Date: Fri Jan 24 03:42:26 +0900 2020 Row: 1 Page: 1

図 4　論理和

Entity:ALU_vlg_tst Architecture: Date: Fri Jan 24 03:32:34 +0900 2020 Row: 1 Page: 1

図 5　排他的論理和



Entity:ALU_vlg_tst Architecture: Date: Fri Jan 24 03:44:33 +0900 2020 Row: 1 Page: 1

図 6　比較演算

Entity:ALU_vlg_tst Architecture: Date: Fri Jan 24 03:47:53 +0900 2020 Row: 1 Page: 1

図 7　移動演算

Entity:ALU_vlg_tst Architecture: Date: Fri Jan 24 03:49:48 +0900 2020 Row: 1 Page: 1

図 8　オーバーフロー


/ALU_vlg_tst/eachvec
/ALU_vlg_tst/dataA
1111111111011010
/ALU_vlg_tst/dataB
0000000001011010
/ALU_vlg_tst/opcode
0000
/ALU_vlg_tst/cond
00x1
0011
/ALU_vlg_tst/export
0x00...
0000000000110100
16800 ps
17000 ps
17200 ps
17400 ps
17600 ps
Entity:ALU_vlg_tst Architecture: Date: Fri Jan 24 03:53:41 +0900 2020 Row: 1 Page: 1

表 1　順序回路の入力と遷移, 出力

Q_0	Q_1	x	Q_0^+	Q_1^+	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1

2.1　期待される機能

入力列 $x_0x_1...x_n$ に対して、$x_ix_{i+1}x_{i+2}x_{i+3} = 1001$ という入力が含まれる場合、最後の 1 を受け取った時に 1 を出力する。今回受理する語を 1001 としたのに特に理由はないが、このような順序機械と同様に様々な入力を検出する順序機械が作れ、0,1 の組み合わせからなる語に適当に言語を割り当てておけば、ある言葉が発せられた時に 1 を出力するような順序機械が作れる。また、この順序機械では入力の順番が大事なので 1,0 の組み合わせからなるパスワードを受理する機械として応用することもできる。

2.2　シミュレーションによって確認できた機能

上の表は論理式表現にすると、

$$Q_0^+ = (Q_0 \oplus Q_1)x^{'} \tag{1}$$

$$Q_1^+ = Q_0Q_1^{'} + x \tag{2}$$

$$y = Q_0Q_1x \tag{3}$$

と表せる。図 1 に示した波形シミュレーションだけではわかりにくいので、各サイクルでの x の値を以下にまとめる。

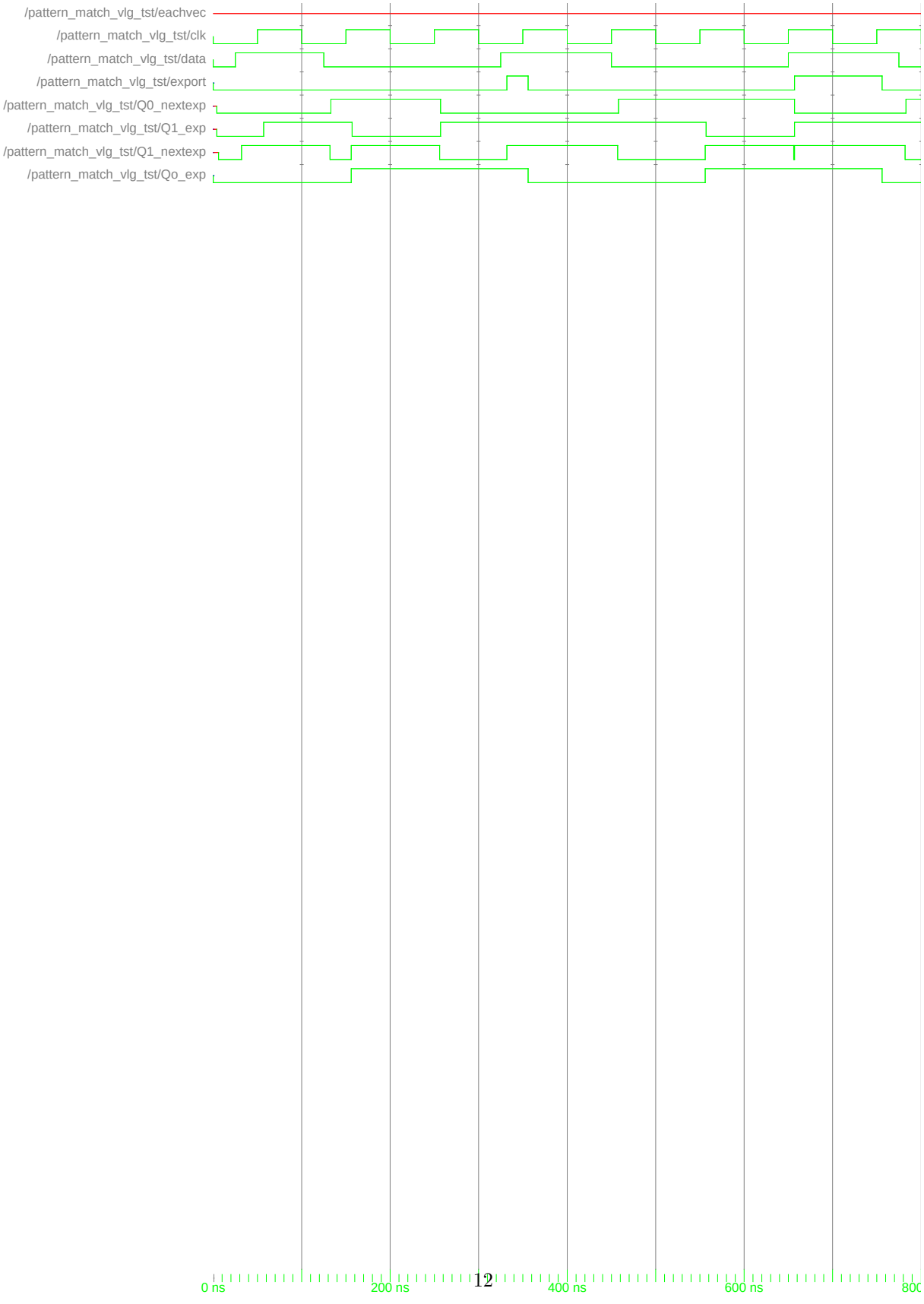
入力:$1 \to 0 \to 0 \to 1 \to 0 \to 0 \to 1 \to 0$

これと図の状態遷移及び出力の変化から状態遷移と出力に関して、真理値表及び、論理式に合致した波形であることが読み取れる。すなわち出力が 1 となるのは 1001 が入力されたときであることがわかる。

2.3　クロック周波数についての考察

Compilation Report から読み取れた情報として、Fmax が 1123.6MHz であった。これはクロックがとる値として可能な最大の値を示している。この値が大きい程 1 サイクルを高速にすることができる。これよりも速い周波数で動作させてしまうと、回路の遅延のために変化が伝わる前に次の素子へと情報が流れてしまい誤動作の原因となる。

図 9　順序機械のシミュレーション



2.4　回路規模の考察

使った論理素子の数について Compilation Report から 3 / 28,848 (≤ 1 %) という値を得た。これは十分に小さい値であると考えられる。今回は作った回路が小規模であるので遅延時間もそんなに問題にならなかったと思われるが、回路の大きさが k 倍になった時に桁上げ伝播加算器を用いた ALU と桁上げ先見加算器を用いた ALU では桁上げ伝播方式では k 倍の計算量がかかるが、桁上げ先見方式では $1 + \frac{\log k}{\log n}$ 倍で抑えられるので今回のように回路規模が大きくならないような回路の場合、桁上げ先遣方式を使ったほうが良いと思われる。

3　参考文献

Computer Organization and Design MIPS Edition, Fifth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)

https://ocw.kyoto-u.ac.jp/ja/09-faculty-of-engineering-jp/computer-architecture-basics/pdf/slide6.pdf