# 平成 28 年度 2 回生後期学生実験 HW レポート

## 計算機科学実験及演習２

## （CAD ツール上での論理設計）

## 第５回、第６回、第７回

締め切り
2017 年 1 月 31 日

提出
2017 年 1 月 30 日

実験者

1029272870

谷　勇輝

# 演習課題１　ALU の設定

次の外部仕様を満たす 16 ビットの ALU(算術論理演算器)を設計しなさい。

- 入力信号
  - dataA[15..0]: 16 ビットの入力値 A
  - dataB[15..0]: 16 ビットの入力値 B
  - opcode[3..0]: 4 ビットの操作コード
- 出力信号
  - out[15..0]: 16 ビットの演算結果
- 機能仕様

右の表に示す出力を行う。

| 操作コード | 機能 | 説明 |
|---|---|---|
| 0000 | out=A+B | 加算(ADD) |
| 0001 | out=A-B | 減算(SUB) |
| 0010 | out=A&B | 論理積(AND) |
| 0011 | out=A\|B | 論理和(OR) |
| 0100 | out=A^B | 排他和(XOR) |

## 1.1　実験方法

上記仕様を満たすための回路の構成方式を複数種類考え、その中で最尤と考えられた回路を CAD ツール QuartusⅡ上で設計、シミュレーションした。

## 1.2　検討

### 1.2.1　減算の実現方式

減算の実現方式として以下に示す３つの回路方式を検討し、それぞれ利点と欠点を比較、最終的には３の方式を採用した。

1. 半減算器と全減算器を構成する方式
2. ２の補数器を構成し、加算器を使って減算を実現する方式
3. 加算器と２の構成を融合し、加算・減算器を構成する方式

#### １の方式

半減算器、全減算器を加算器と同様に作成する。最下位ビットの計算に半減算器を、それ以外のビットの計算に全減算器を充て、それらを連結させ減算器を構成する。

「減算器」という**独立した素子**ができ、ALU として纏める際に最も**直感的に設計ができる**という点に利点を見出すことが出来る。一方で新たなものを新しく作るため**必要工数が跳ね上がり、コストも多くかかってしまう**ことが予想される。また、独立した素子を作るため、使用する**総基本素子数も大きくなってしまう**。

**２の方式**

Ｘの２の補数を$X'_2$と表すこととすると、減算は以下のように変形できる。

$$\mathrm{A}-\mathrm{B}=\mathrm{A}+B'_2$$

従って、２の補数器を作成すれば、減算を既存の加算器を利用して実現できる。

減算を実現するまでに使用する素子は、２の補数器と加算器２つなので、新たに作成する素子は２の補数器のみである。従って、１の方式に比べ**工数を小さく抑えられる**。しかし一方で、結局加算器を２回使用しているため**総素子数はあまり変わらない**。

**３の方式（採用）**

２の方式で減算の中に加算器を使用していることに着目し、加算器に減算器の機能を追加する方針を立てた。

Ｘのビット反転（１の補数）をX′と表すと、減算は、

$$\mathrm{A}-\mathrm{B}=\mathrm{A}+\mathrm{B}'+1$$

と表すことができる。

２の方式ではこの＋１をわざわざ加算器を１回使って実現していたが、加算器の最下位ビットの計算に充てる部分を半加算器から全加算器に変更することでこの＋１を実現し、総素子数の削減を図る（図１）。

加算ならばそのままのＢを、減算ならばB’を第二引数として加算器に取り入れる必要があるため、新たにマルチプレクサ（図２）を設計する。また、加算・減算のどちらの機能を実行するかを制御するため、１ビットの制御信号を引き入れる。加算に０、減算に１を割り当て、マルチプレクサと加算器の最下位ビットの全加算器にこの制御信号を送れば、加算・減算器が実現できる（図３）。

この方式では、減算を実現する為に新たに設計した素子はマルチプレクサのみであり、**工数とコストの大幅削減が見込める**。また、加算器と素子を共有するため**総素子数を半減させる**ことができる。減算を行う際に使用される加算器も１つのみであるので、２の方式に比べ**処理速度の向上**も見込まれる。

### 1.2.2　操作コードによる機能変更の実現方式

操作コードによる機能切り替えを実現する方式として、以下の２つの回路方式を検討、利点と欠点を比較し、最終的に２の回路方式を採用した。

1. AND 回路によって選択した以外の出力を０にし、選択した出力のみを残すことで実現する方式
2. マルチプレクサによって必要な出力をトーナメント方式で選び取っていく方式

**１の方式**

AND 回路の一方の入力を制御信号とすることで、出力を有効にするか無効にする（０にする）かを制御できる。それぞれの機能を実現する回路の出力に対し有効・無効の制御処理を行い、それらを AND 回路で束ね、最終的な出力とすることで切り替えを実現する。

この方式で難しいのは、制御信号の作成である。操作コードに対応した機能の出力のみを有効とし、それ以外を無効とするような制御信号を再構築する必要がある。

この方式の特徴として、制御のために主要回路に追加される素子は少ない（１機能に対し２回 AND 回路を通れば良い）一方で、制御信号の作成のために多くの素子を使用する必要がある点が挙げられる。操作コードのアーキテクチャをこの方式に合うようにあらかじめ設計することで幾分かの負担軽減にはなるが、選択肢が多くなるとそれも限界が生じてしまう。

**２の方式（採用）**

マルチプレクサ（図１）は、２入力のうち片方を選び取るための素子である。これをトーナメント方式で複数段に渡って配置し、最終的に１つの出力を選び抜くことで機能の切り替えを実現する。

１の方式が機能の種類ｎに関わらず常に AND 回路２段で構成されるのに対し、こちらは$log_2 n$段を必要とするため、選択する機能が多くなれば多くなるほど段数がかさみ、処理速度が落ちてしまう欠点がある。一方で操作コードが２進数の連番で表されている場合、操作コードの下位第ｋ番目のビットをそのまま第ｋ段目のマルチプレクサの制御信号に使用することができる。結果、設計がシンプルなものになるため、工数を抑えられることが期待できる。

## 1.3　設計回路

設計した回路を図４に示す。また、回路中に使用したプリミティブでな

い回路は以下の通りである。

- 16MUX
  - ・16 ビットバス信号用マルチプレクサ。
  - ・図２を 16 ビット入力用に拡張したもの。
- 16AddSub
  - ・1.2.1 の３の方式で検討した加算・減算器。（図３）
  - ・以下に示す 8fullAdder を２つ使用し、16 ビット加算器を実現し、16MUX でその入力を選択している。
- 8fullAdder
  - ・8 ビット加算器（図１）。最下位への繰り上がり入力あり。
  - ・fulladder_bus はバスを入力とした全加算器。
  - ・8x2to2x8 はバスの各ビットごとのまとめ直しをする。
  - ・16x1to1x16 も同様のまとめ直しを行っている。

## 1.4　シミュレーション

### 1.4.1　入力と出力

dataA、dataB に様々な値を代入し、そのそれぞれに対して５種類の機能が全て正常に動作していることを確認した。dataA、dataB の値は、各桁の複合条件網羅になるよう、また全加算器、AND 素子、OR 素子、XOR 素子それぞれの複合条件網羅となるよう設定した。設定と結果を以下の表１に示す。

表 1　ALU シミュレーションの入出力．全て正常に動作している．

| dataA | dataB | opcode | out | 正 |
|---|---|---|---|---|
| 1111000000001111 | 1111000011110000 | 0000 | 1110000011111111 | ○ |
| | | 0001 | 1111111100011111 | ○ |
| | | 0010 | 1111000000000000 | ○ |
| | | 0011 | 1111000011111111 | ○ |
| | | 0100 | 0000000011111111 | ○ |
| 0000000011111111 | 0000111100001111 | 0000 | 0001000000001110 | ○ |
| | | 0001 | 1111000111110000 | ○ |
| | | 0010 | 0000000000001111 | ○ |
| | | 0011 | 0000111111111111 | ○ |
| | | 0100 | 0000111111110000 | ○ |

| | | | | |
|---|---|---|---|---|
| 0000111111110000 | 1111000011110000 | 0000 | 0000000011100000 | ○ |
| | | 0001 | 0001111100000000 | ○ |
| | | 0010 | 0000000011110000 | ○ |
| | | 0011 | 1111111111110000 | ○ |
| | | 0100 | 1111111100000000 | ○ |
| 1111111100000000 | 0000111100001111 | 0000 | 0000111000001111 | ○ |
| | | 0001 | 1110111111110001 | ○ |
| | | 0010 | 0000111100000000 | ○ |
| | | 0011 | 1111111100001111 | ○ |
| | | 0100 | 1111000000001111 | ○ |

### 1.4.2　伝搬遅延時間と回路規模

- 最大伝搬遅延時間　　35.744 (ns)（opcode[0] -> out[15]）
- 回路規模　　55 （論理素子数　Total Logic Elements）

## 1.5　考察

1.2 節での検討により回路規模についてはかなり抑えることができている。一方、遅延時間についてはあまり考慮して設計できなかった。実際、ハードウェアに詳しい『コンピュータの構成と設計 第５版』 [Patterson Hennessy, 2014]ではパイプライン処理の説明の際に ALU の処理速度を 200ps としており、今回設計した ALU よりも 180 倍近く高速であることを示唆している。

伝搬遅延時間を短縮する方法について考察する。現在の設計で最も遅延に影響を与えていると考えられるのは**加算器**である。現在の加算器の設計は全加算器を直列に 16 個配置しているため効率が悪い。桁上げ先読みなどを行うことによって遅延時間の大幅短縮を図ることができる。

さらに桁上げ先読みを行う場合、各入力ビットごとの論理和、論理積、各出力ビットごとの排他的論理和が必要となるため、それらを使用して加算と減算以外の残りの３つの機能についても統合を図ることができると考えられる。

このように設計方式を変更することで、回路規模の増加を最小限に抑えつつ、遅延時間の大幅短縮を行うことができる。

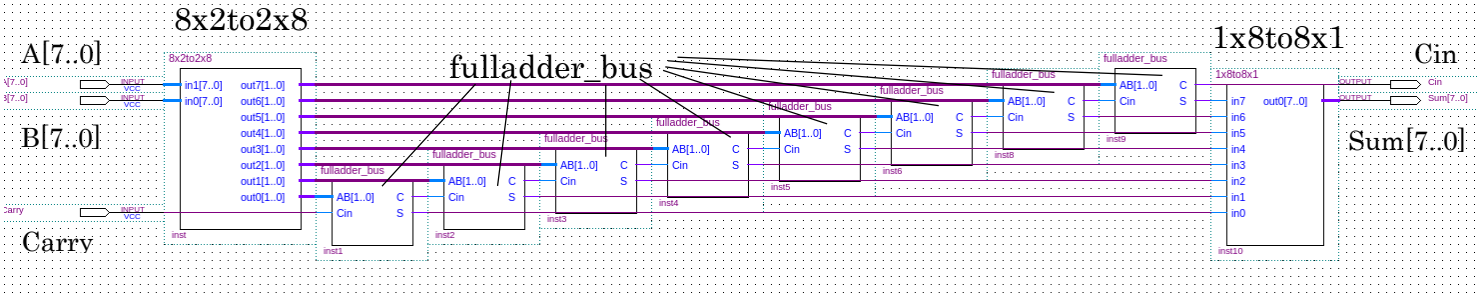


図 1　8 fulladder　　最下位ビットへの繰り上がりを入力としてもつ 8 ビット加算器

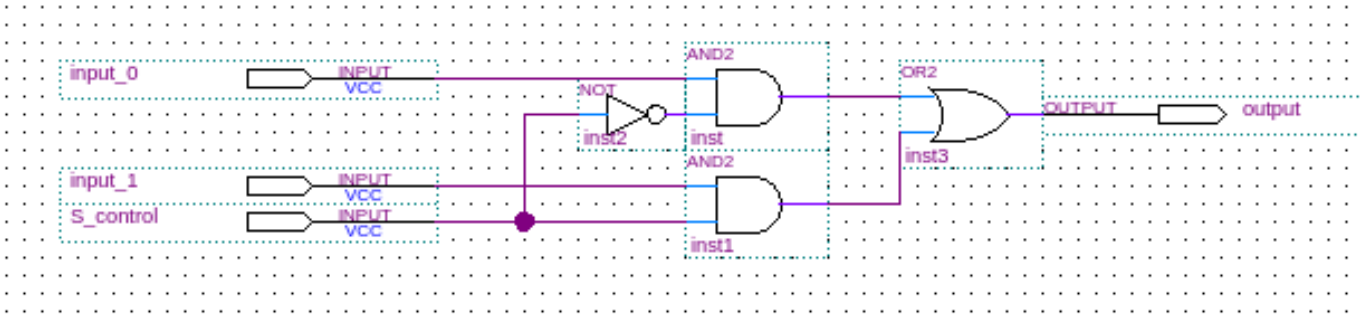


図 2　MUX　　マルチプレクサ（選択器）

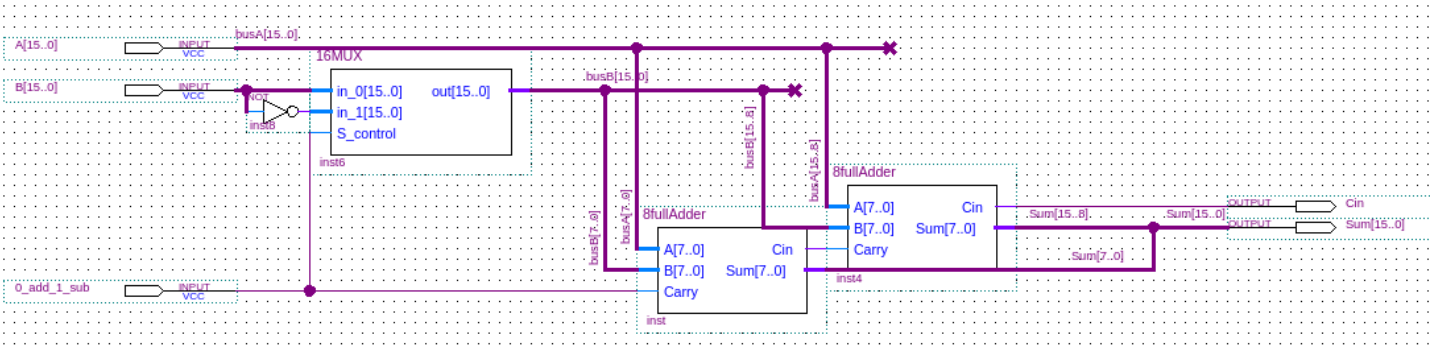


図 3　16AddSub　　16 ビット加算・減算器.

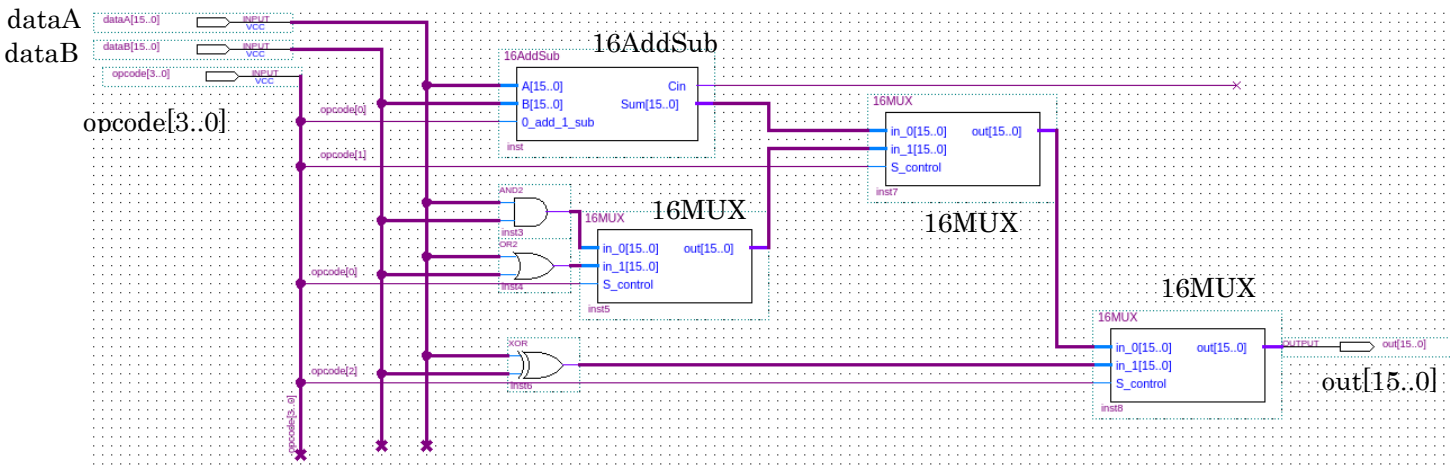


図 4　ALU　　5 つの機能を MUX によって選び取る

# 演習課題２　カウンタの設計

次の外部仕様を満たす８ビットの同期式カウンタを設計しなさい。

- 入力信号
  - clock: クロック信号
  - reset: リセット信号
  - amount[3..0]: 4 ビットのカウント量
  - direction: カウント方向
  - maxcnt[7..0]: 8 ビットのカウンタの最大値
  - mincnt[7..0]: 8 ビットのカウンタの最小値
- 出力信号
  - count[7..0]: 8 ビットのカウント値
- 機能仕様
  - 同期式順序回路である。
  - クロック信号の立ち上がりに応じて amount ずつカウントアップする。
  - reset が 1 である時は、カウント値を 0 としてカウントアップしない。
  - カウント値がオーバーフローした場合は 0 に戻る。
  - カウント値がアンダーフローした場合はカウンタの最大値に戻る。
  - direction が 0 の場合はカウントアップ、1 の場合はカウントダウンする。
  - カウントアップ中にカウント値が maxcnt を超えた場合は 0 に戻る。
  - ~~カウントアップ中時のカウントアップ値は mincnt から開始する。~~
  - カウントダウン中にカウント値が mincnt を下回った場合は maxcnt に戻る

## 2.1　実験方法

上記仕様を満たすための回路の構成方式を複数種類考え、その中で最尤と考えられた回路を CAD ツール Quartus Ⅱ 上で設計、シミュレーションした。

発展機能も実装を試み、カウントアップの初期値設定（上記仕様で取り消し線の入った仕様）以外の実装には成功した。

ここでのアンダーフローは「負のオーバーフロー」を指すと解釈し、その場合の「カウンタの最大値」は maxcnt の事だと解釈して実装を行った。

## 2.2　検討

### 2.2.1　オーバーフロー・アンダーフローの検出の実現方式

オーバーフロー、アンダーフローの検出方法として、以下に示す２つの回路方式を検討し、それぞれ利点と欠点を比較、最終的には２の方式を採

用した。

1. 加算器の２入力の最上位ビットと出力の最上位ビットで判定する方式
2. 加算器の上位２ビットの繰り上がりで判定する方式

**１の方式**

加算のオーバーフローとアンダーフローは、以下の方法で判定できる。

オーバーフロー：正の数＋正の数＝負の数となった時

アンダーフロー：負の数＋負の数＝正の数となった時

２の補数表現では最上位ビットが正負を表すため、２つの入力と出力の最上位ビットを使用すればよい。

この方式の利点は、ALU を作成の際に作成した加算器をそのまま流用できるという点である。欠点としては３つの信号について比較する必要がある点、減算の場合、反転し１足したものを入力として扱う必要がある点が挙げられる。特に後者について、ALU の際に使用した加算・減算器では＋１を入力とは別に行っているため相性が悪い。

**２の方式（採用）**

加算のオーバーフローとアンダーフローは、以下のビットの繰り上がりの有無によって判定できる。

オーバーフロー：上位第１ビット　なし　第２ビット　あり

アンダーフロー：上位第１ビット　あり　第２ビット　なし

この方式の欠点は加算器の第２ビットでの繰り上がりを新たな出力として引き出す必要がある点である。利点としては比較対象が２ビットのみである点、加算・減算の区別が必要ないという点が挙げられる。

## 2.3　設計回路

設計した回路を図５に示す。回路中に使用したプリミティブではない回路は以下の通りである。

- 8MUX
  - ・8 ビットマルチプレクサ。図２の８ビット版

- 8gthan
  - ・8 ビット比較演算器。（図６）
  - ・A>B の時に１、それ以外は 0 を出力
  - ・内部的には減算 B-A を行い、負かアンダーフローで１を出力

D-FF がカウント値を保持し、次のクロックが立ち上がるまでにカウント値と amount で加算・減算を行う。３つのマルチプレクサはそれぞれ、①reset が 1 の時に結果を 0 とする機能、②オーバーフローまたは maxcnt を上回った場合の結果を 0 とする機能、③アンダーフローまたは mincnt を下回った場合の結果を maxcnt とする機能を実現している。

## 2.4 シミュレーション

### 2.4.1 入力と出力

全ての機能が正常に機能していることを確認できるよう、以下のように入力信号を定めてシミュレーションを行った。表 2 に示すよう各値を遷移させ出力が正しいことを確認した。表 2 に示した入力以外の入力は以下の通りである。

- clock　　50ns 毎に反転するクロック信号
- maxcnt　固定値 00100000　　10 進で 32
- mincnt　固定値 11100000　　10 進で-32

なお、入力の変更は全てクロックの立ち上がりの 50ns 前に行った。

アンダーフロー、オーバーフローについては maxcnt、mincnt の検出を行っている状態については不必要であるが、これも別途確認を行い正常動作を確認した。

**表 2　カウンタのシミュレーションの入出力．機能は全て正常に動作している．**

| クロック数 | amount | direction | reset | count | 正 |
|---|---|---|---|---|---|
| 1 | 0011 (3) | 0 (＋) | 0 | 00000000 (0) | ○ |
| 2 | | | | 00000011 (3) | ○ |
| 3 | | | | 00000110 (6) | ○ |
| 4 | | | | 00001001 (9) | ○ |
| 5 | 1010 (10) | | | 00010011 (19) | ○ |
| 6 | | | | 00011101 (29) | ○ |

| | | | | | |
|---|---|---|---|---|---|
| 7 | | 1 (−) | | 00010011 (19) | ○ |
| 8 | | | | 00001001 (9) | ○ |
| 9 | | | 1 (reset) | 00000000 (0) | ○ |
| 10 | | 0 (+) | 0 | 00001010 (10) | ○ |
| 11 | | | | 00010100 (20) | ○ |
| 12 | | | | 00011110 (30) | ○ |
| 13 | | | | 00000000 (0) | ○ |
| 14 | | 1 (−) | | 11110110 (-10) | ○ |
| 15 | | | | 11101100 (-20) | ○ |
| 16 | | | | 11100010 (-30) | ○ |
| 17 | | | | 00100000 (32) | ○ |
| 18 | | | | 00010110 (22) | ○ |
| 19 | | | | 00001100 (12) | ○ |
| 20 | | | | 00000010 (2) | ○ |
| 21 | | | | 11111000 (-8) | ○ |

### 2.4.3　性能と回路規模

タイミング制約としてクロックを 10ns（5ns-1,5ns-0 のステップ信号）に設定して以下の最大周波数を測定した。

- 最大周波数　133.05MHz
- 最大 setup 時間　11.291ns
- 回路規模　38　（総素子数　Total logic elements）

## 2.5　考察

回路は期待した通りに動作し、発展機能も実装したものについては実現できた。回路規模も 38 と比較的コンパクトに抑えることができている。

実装できなかった初期値の発展機能について考察する。D-FF にはプリセット端子とクリア端子があり、それらに適切な入力をすることで非同期的に D-FF の内容を強制書き換えすることができる。第 1 クロックを「準備クロック」と定義し直して良いのならば、1 回目のクロックを感知するまでは mincnt を、してからは 1 を出力する回路を構成することで、初期値の設定を行うことができた。

その他にも、分周器を使いクロックを分割する方法、第 1 クロックのみ amount を mincnt に切り替えて入力する方法などが考えられたが、時間の関係上今回の実験では実装するに至らなかった。

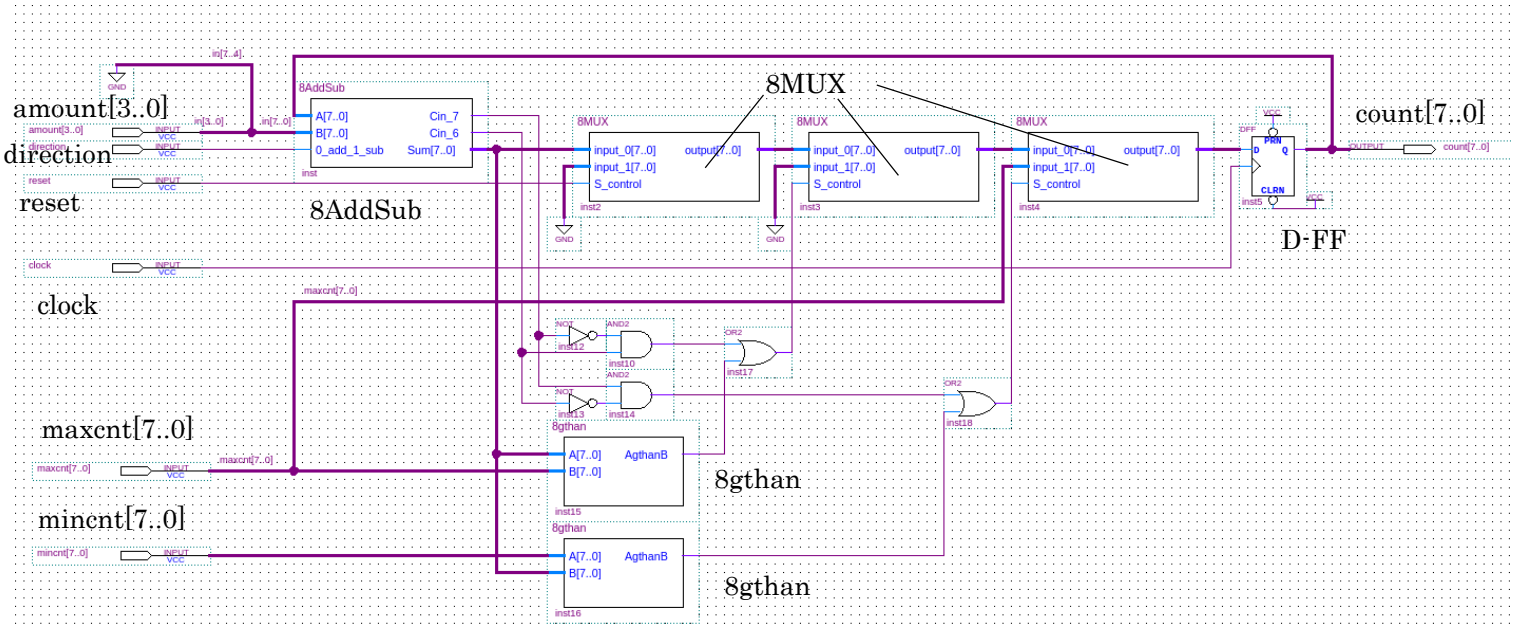


図 5　　counter

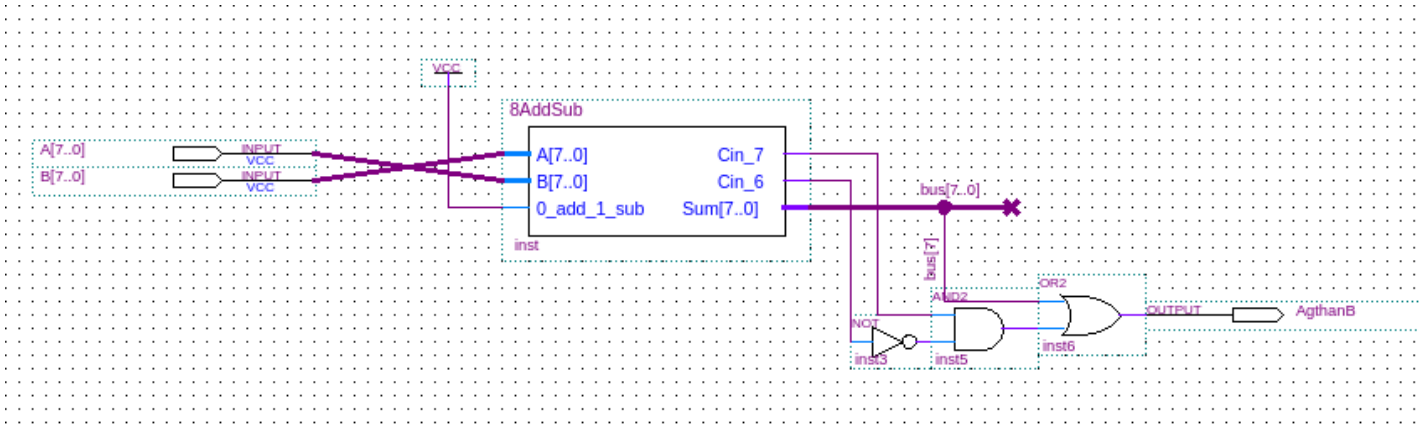


図 6　　8gthan　　比較演算器（A>B）

# 演習課題３　シフタの設計

次の外部仕様を満たす 16 ビットのシフタを設計しなさい。

- 入力信号
  - ~~input~~[15..0]: 16 ビットの入力データ
  - amount[3..0]: 4 ビットのシフト量
  - opcode[3..0]: 4 ビットの操作コード
- 出力信号
  - out[15..0]: 16 ビットのシフト結果
- 機能仕様
  - 右の表に示す出力を行う。

| 操作コード | 機能 |
|---|---|
| 1000 | 左論理シフト（０埋め） |
| 1001 | 左循環シフト |
| 1010 | 右論理シフト（０埋め） |
| 1011 | 右循環シフト |

## 3.1　実験方法

上記仕様を満たすための回路を CAD ツール QuartusⅡ上で設計、シミュレーションした。

入力信号 input[15..0]について、input という名称がシミュレーションツールを使用する際のテストベンチファイルの記入方式の予約語にあたり、このままではシミュレーションが上手く動かなかったので in[15..0]という名称に変更して設計を行った。

## 3.2　設計回路

図 7 に設計した回路を示した。回路中で使用したプリミティブでない回路については以下の通り。

- 16MUX
  - ・16 ビットマルチプレクサ。
  - ・ALU の作成時と同じもの。

シフト演算自体は、バスの内部の単線を並び替える、或いは 0 とすることで実現できる。機能の切り替えも ALU の 1.2.2 で検討した 2 の方式に習いマルチプレクサを層状に配置することで同様に実現できる。

問題は amount ビットシフトする機能の実現である。これは単純に 16 種類の単線並び替えを用意しても良いが、4 種類の機能と合わせて 64 の単線並び替え用回路を用意する必要があり、工数がかかる。そこで、二進法の考え方を利用し、16 までの数字を 4 種類のシフトの組み合わせで実現することを考えた。８ビッ

ト、４ビット、２ビット、１ビットのシフト回路を作り、マルチプレクサでそのシフトを行うか否かの選択を行っている。この時、制御信号として amount の各ビットをそのまま使用でき、回路全体もシンプルにまとまった。

## 3.3 シミュレーション

### 3.3.1 入力と出力

以下の表３に示す入力を行い全ての機能が正常に機能していることを確認した。各機能と４種類のシフタに対して判定条件網羅となるように入力パターンを簡潔に作成した。

**表 3 シフタのシミュレーションの入出力. 機能はすべて正常に動作している.**

| in[15..0] | amount[3..0] | opcode[3..0] | out[15..0] | 正 |
|---|---|---|---|---|
| 1000110011101111 | 1010 (10) | 1000 | 1011110000000000 | ○ |
| | | 1001 | 1011111000110011 | ○ |
| | | 1010 | 0000000000100011 | ○ |
| | | 1011 | 0011101111100011 | ○ |
| | 0101 (5) | 1000 | 1001110111100000 | ○ |
| | | 1001 | 1001110111110001 | ○ |
| | | 1010 | 0000010001100111 | ○ |
| | | 1011 | 0111110001100111 | ○ |

### 3.3.2 性能と回路規模

- 最大伝搬遅延時間　22.234 (ns)　（amount[3] -> out[11]）
- 回路規模　122　(総素子数 total logic elements)

## 3.4 考察

回路の設計はシンプルで簡潔なものとすることができた。ALU で使用した素子を再利用したものしか使用していないため工数も小さく、短時間で設計することができた。

最大伝搬遅延時間はシフタの種類を増やし並列にすればするほど短くすることができる。究極的には 16 種類の単線並び替えを全て並列すれば１段で構成することができるが、その分素子の数が増加し回路規模が大きくなってしまう。

今回の設計ではバランスの良い４種類４段で構成し回路規模を小さくしよう

と試みたが、それでも回路規模は 122 と ALU の約 2 倍にもなってしまった。単線の統合なども「素子を使用している」扱いになると考えられる。この遅延時間を短縮するためには MUX の改良が必要になるだろう。

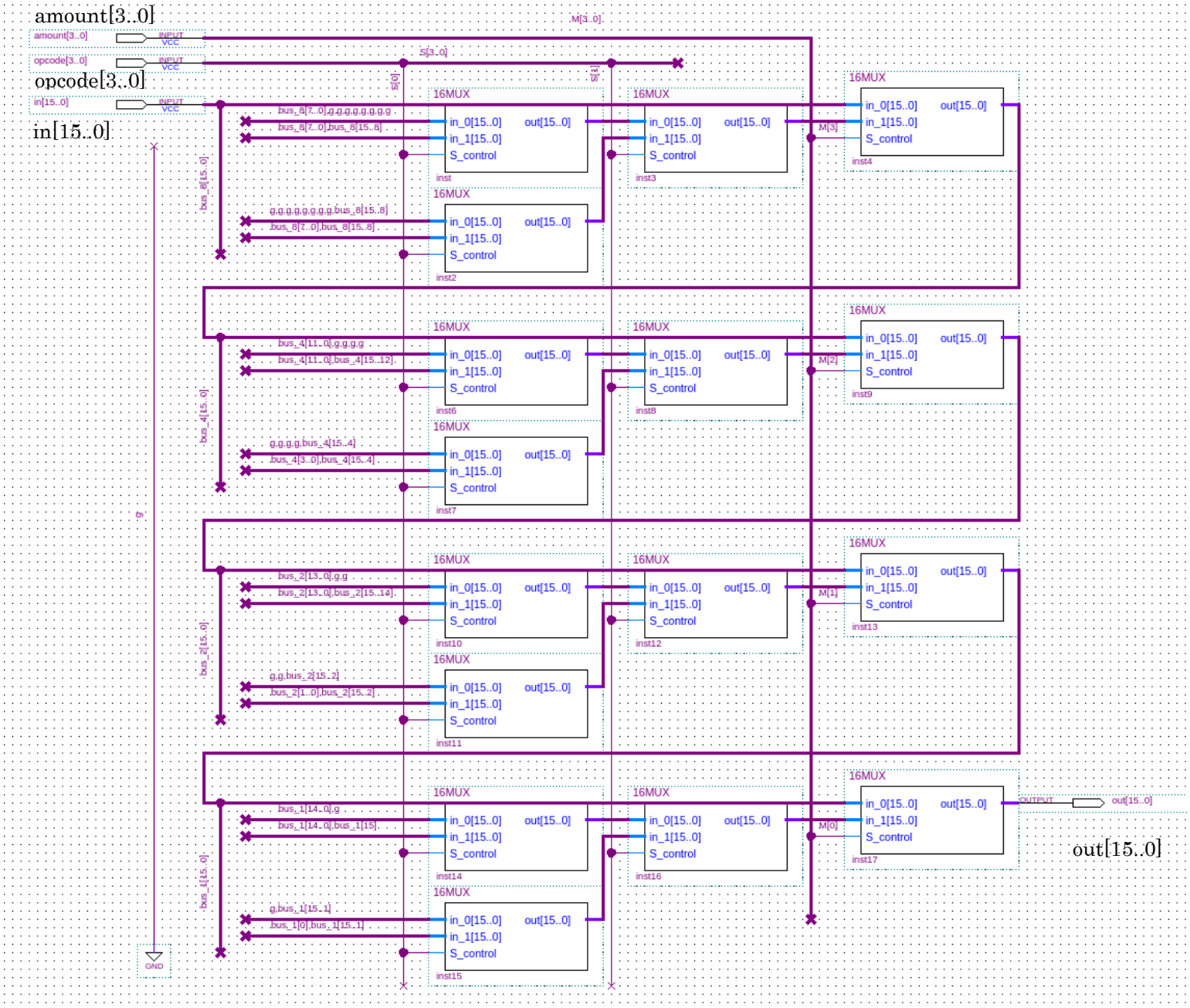


図 7　shifter　上から 8bit,4bit,2bit,1bit のシフトを担当するブロック.

# 調査課題

## 4.1　調査課題 1

| |
|---|
| FPGA の最近の実用例について調べ、その詳細をまとめなさい。 |

FPGA はいわゆる「ムーアの法則」に従い集積度と性能が指数関数的に向上し、近年様々な用途で利用されるようになってきた。

FPGA は近年の宇宙開発において重要な立ち位置にある。2017 年 1 月 14 日、人工衛星群「Iridium NEXT」の第一弾となる 10 機の打ち上げが行われた。この人工衛星には、宇宙空間での長期使用に耐えうる耐久性、低消費電力、柔軟性を備えた最新鋭の航空宇宙グレード FPGA が搭載されている。この FPGA により、Iridium NEXT は軌道中にありながら進化していく先々の技術を随時導入し稼動することが可能になっているのである。

また、情報処理の高速化の観点でも FPGA は貢献している。例えば、2016 年 12 月 1 日に発表されたミラクル・リナックス株式会社の文字列分割処理の高速化についての研究開発成果によれば、FPGA と独自研究した追加のフレームワークによって、テキストログのバッチ処理を想定したベンチマークにおいて CPU 利用時の 10 倍の高速化に成功したとされている。

総じて FPGA は現在も高集積化、低価格化の進化を続けているといえる。FPGA 上に CPU コアを搭載するといった流れのみならず、原理的に動作中にも再構成可能であることから「再構成可能コンピューティング」という新たなコンピュータのアーキテクチャも生まれた。FPGA はその柔軟性から、幅の広い、多角的な発展を続けている。

## 4.2　調査課題 2

| |
|---|
| ハードウェア記述言語（HDL）について調べ、その詳細をまとめなさい。 |

ハードウェア記述言語（HDL）は、ハードウェア、主に集積回路の構成や動作仕様を記述するコンピュータ言語である。ハードウェアの基本属性である時間や並行性を記述するものであり、プログラム言語とは類似する点もあるが、根本的に異なるものである。

プログラム言語が処理内容の機械語を生成する道具とするならば、HDL は回路の接続関係を記述するネットリストと呼ばれる言語を生成する道具と言える。プログラム言語のコンパイラにあたる変換器は HDL ではシンセサイザと呼ばれる。

主なものには、米国国防総省の納入書に一端をもつ VHDL（IEEE1076)と、民間企業の商用シミュレーションツールから生まれた Verilog HDL(IEEE1364)があり、いずれも抽象的な記述から具体的な接続まで記述できる汎用言語である。

## 参照文献


PattersonADavid, HennessyJohnL. (2014). コンピュータの構成と設計 第5版（上）. 日経 BP 社.

ザイリンクス、「Iridium NEXT」衛星群に数百個の航空宇宙グレード FPGA が搭載. (日付不明). 参照先: 組み込みシステム技術者向けマガジン E.I.S: http://www.eis-japan.com/release/20170124/

ハードウェア記述言語. (日付不明). 参照先: FPGA 用語集: http://www.fpga-net.jp/fpga/word/hardware_description_language.html

ハードウェア記述言語. (日付不明). 参照先: wikipedia: https://ja.wikipedia.org/wiki/ハードウェア記述言語