VHDL を用いた簡易プロセッサの試作

仕様書

2018 年 1 月 25 日

A-2 班

15173009　加藤 大登

15173046 久朗津 宏樹

15173088　佐藤 竜郎

15173091　高田 大樹

1 はじめに

本実験では，VHDL を用いた簡易プロセッサの試作を行った．本仕様書では，実際に試作したプロセッサの仕様を記述している．

2 開発環境

OS	…	Windows 8.1
開発ツール,シミュレータ	…	ModelSim-Altera10.3c

3 役割分担

本実験を進めていくうえで行った役割と氏名をまとめたものを以下の表 1 に示す．

表 1．班員の役割

氏名	役割
加藤	仕様立案，制御信号生成回路作成
久朗津	プレゼン資料作成，バス作成
佐藤	仕様書作成，レジスタ作成
高田	FPGA 実装，メモリ作成

4 設計指針

本実験は 0～65535 の整数（16 進数で 0000～ffff）の平方根を求めることができるプロセッサの制作を目的として行った．そのため加減算を行う命令，数値をレジスタに保存する命令，論理演算命令，数値比較命令，ジャンプ命令等が必要となった．

VHDL 記述を簡単にするため命令はすべて 1 ワード命令とした．そのため数値比較命令は減算命令を利用し，論理演算命令は NAND のみとして命令数を削減した．さらに，汎用レジスタの値をプログラムレジスタに格納する命令を追加し，疑似的な関数の作成ができるようにした．

5 プロセッサ仕様

5 章ではプロセッサの仕様について記述する．

5.1 メモリ，汎用レジスタ，フラグレジスタの仕様

今回制作したプロセッサは，メモリ幅を 16bit，深さを 8bit（アドレス：0～255）とした．メモリのアドレスを 0～255 としているのは，全ての命令を 1 ワードとしたため，命令フィールドのアドレス指定部分が 8bit しか確保できないためである．

汎用レジスタは GR0~GR15 までの 16 個で構成している．それらはすべて 16bit のデータを格納する．

フラグレジスタは，演算結果の先頭の bit が 1 であるときに 1 となるサインフラグ S（1bit），演算結果がすべて 0 になったときに 1 となるゼロフラグ Z（1bit），演算結果がオーバーフローしたときに 1 となるオーバーフローフラグ O（1bit）がある．

5.2 プログラムレジスタ，命令レジスタ，MAR，MDR の仕様

すべてのレジスタが 16bit のデータを格納する，ただしアドレス値として扱う場合は，下位 8bit のみを利用する．

プログラムレジスタはカウンタ機能付きで，制御信号に従いデータに 1 を加算することができる．

命令レジスタは一命令終了後にプログラムレジスタが指すメモリのデータを保存する．

MAR，MDR はメモリアクセスに利用する．メモリを参照する際は MAR の値をアドレスとして読み込み，入出力のデータは MDR を経由する．

5.3 命令フィールド

命令フィールドは以下の表 2 に示す通りであり，レジスタ間命令は①，メモリレジスタ間命令や LAD は②の命令フィールドである．

表 2　命令フィールド

①

OP	r1	0000	r2
4	4	4	4

②

OP	r1	addr
4	4	8

5.4 命令セット

以下の表 3 に今回作成したプロセッサの命令セットをまとめたものを記述する.

表 3. 命令セット

命令	機械語	内容
HALT	0000	終了.
LD①	0001 r1 0000 r2	r2 の内容を r1 にコピー.
LD②	0010 r1 addr	メモリアドレスの内容を r1 にコピー.
LAD	0011 r1 addr	メモリアドレスのアドレス値を r1 にコピー.
STR	0100 r1 addr	r1 の内容をメモリアドレスの場所にコピー.
ADD	0101 r1 0000 r2	r1＋r2 の答えを r1 に入れる．FFFF を超えるならフラグ O を 1 にする.
SUB	0110 r1 0000 r2	r1－r2 の答えを r1 に入れる. 負になるならフラグ S を 1 にする.
SL	0111 r1 0000 r2	r1 のビットを r2 の内容分だけ左へ動かす．最後にあふれた値をフラグ O に入れる.
SR	1000 r1 0000 r2	r1 のビットを r2 の内容分だけ右へ動かす．最後にあふれた値をフラグ O に入れる.
NAND	1001 r1 0000 r2	r1，r2 の否定論理積の答えを r1 に入れる.
JMP	1010 r1 addr	メモリアドレスのアドレス値をプログラムレジスタにコピー.
JZE	1011 r1 addr	フラグ Z が 1 なら，メモリアドレスのアドレス値をプログラムレジスタにコピー.
JMI	1100 r1 addr	フラグ S が 1 なら，メモリアドレスのアドレス値をプログラムレジスタにコピー.
JOV	1101 r1 addr	フラグ O が 1 なら，メモリアドレスのアドレス値をプログラムレジスタにコピー.
RJMP	1110 r1 0000 r2	r1 の内容をプログラムレジスタにコピーした後，プログラムレジスタの値を 1 増加する.
DISP	1111 r1 0000 r2	r1 の下 4bit を r2 で指定する 7 セグメント LED に出力.

※JMP，JZE，JMI，JOV は r1 に 0 以外が指定されたとき，r1 にジャンプ前のアドレスを書き込む.

5.5 制御信号生成回路

設計したプロセッサでは，命令レジスタのデータを制御信号生成回路が命令として解釈し，いくつかのマイクロ命令に分けて実行している．一つの命令を完了するのに 6〜9 クロックかかるため，内部に状態を保存する領域を持っていて，クロックごとにマイクロ命令を切り替えている．

制御信号生成回路からの出力を以下の表 4 に示す．

表 4．制御信号生成回路の出力

信号名	bit	説明
ba_ctl	3	バス A への入力データを選択する
bb_ctl	5	バス B への入力データを選択する
address	8	命令が含むアドレス値（命令フィールド②）
gr_lat	1	汎用レジスタの書き換え可能フラグ
gra	4	バス A に出力する汎用レジスタの選択
grb	4	バス B に出力する汎用レジスタの選択
grc	4	書き換える汎用レジスタの選択
ir_lat	1	命令レジスタの書き換え可能フラグ
fr_lat	1	フラグレジスタの書き換え可能フラグ
pr_lat	1	プログラムレジスタの書き換え可能フラグ
pr_cnt;	1	プログラムレジスタのカウンタ機能フラグ
mar_lat	1	MAR の書き換え可能フラグ
mdr_lat	1	MDR の書き換え可能フラグ
mdr_sel	1	MDR への入力データを選択する
m_read	1	メモリの読み込み可能フラグ
m_write	1	メモリの書き込み可能フラグ
func	4	ALU への命令コード
phaseView	4	制御信号生成回路の内部状態

5.6 ALU

制御信号生成回路の命令コード func に従って演算を行い，結果を出力する．演算を行った際にフラグレジスタの更新も行う．

func による処理の内容を以下の表 5 に示す．

表 5．ALU で行う演算

func	ALU での処理内容
0000	バス A の値を出力
0001	バス B の値を出力
0101	ADD，フラグレジスタの値を更新
0110	SUB，フラグレジスタの値を更新
0111	左シフト
1000	右シフト
1001	否定論理積
1010	バス A の値を出力
1011	Z フラグが 1 のときバス A，0 のときバス B+1 を出力
1100	S フラグが 1 のときバス A，0 のときバス B+1 を出力
1101	O フラグが 1 のときバス A，0 のときバス B+1 を出力
1110	バス A の値を出力
1111	バス A の下 4bit を出力

5.7 回路図

プロセッサを作成するための概略図である簡単な回路図を以下の図 1 に示す.

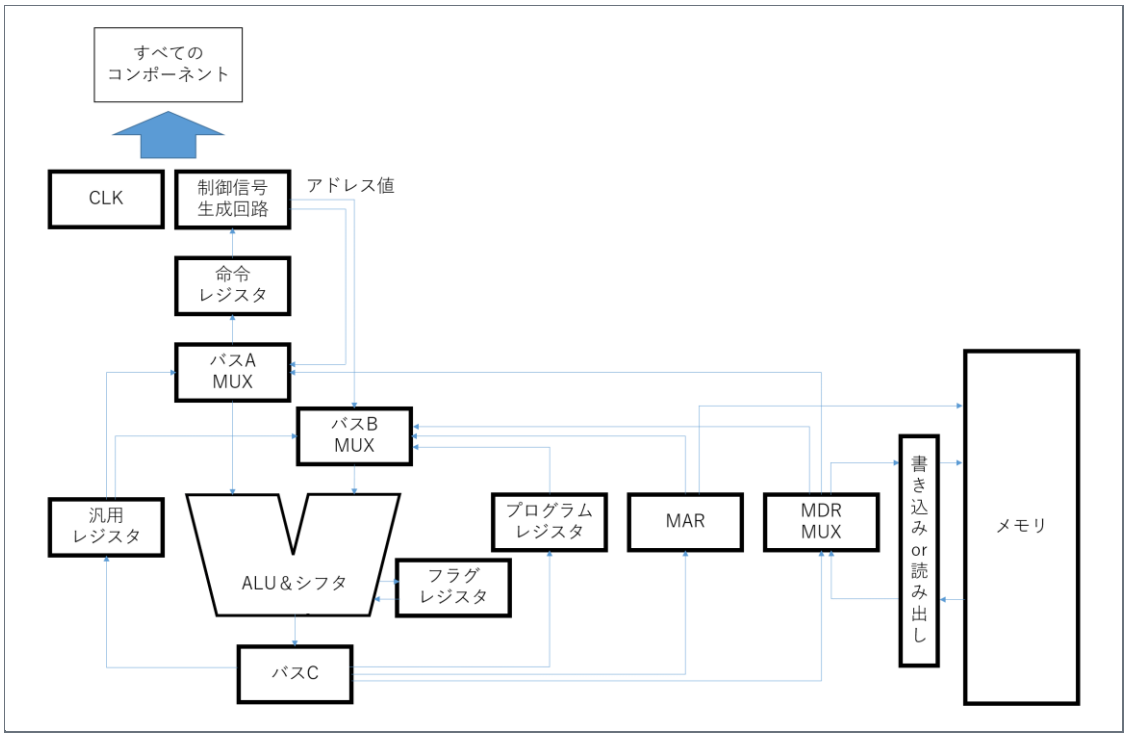

図 1. 回路図

6 テスト

各コンポーネントを結びつけるコンポーネント core を作成し，それをテストベンチとした．メモリの初期状態はテキストファイルに記述し，実行時に読み込むようにしている．

6.1 テストプログラム（Python）

当初の目的通り，平方根の計算を行うプログラムを作成した．アルゴリズムが確実に動作するかを確認するために，まず Python でアルゴリズムを実装した．コードは以下の図 2 に示す．

```
MOD = 2 ** 16
def func0():
    if j < 0:
        return x1 << -j
    else:
        return x1 >> j
def func1():
    if j < 0:
        return y >> -j
    else:
        return y << j

x1 = int(input(), 16)
x0 = 0
a = 0
y = 0
n = 0
c = 0
t = x1

while t > 0:
    t >>= 1
    n += 1
n += 16
n += n & 1
```

```
print(hex(x1), '=', x1)

for i in range(n, -1, -2):
    j = i - 16
    y21 = 0
    a <<= 1
    y <<= 1
    if y >= MOD:
        y %= MOD  # 下 16 ビットをとる
        y21 += 1
    y20 = 1 | y
    c = True
    f0 = func0() % MOD
    x20 = x0 >> i
    x20 += f0
    x21 = x1 >> i
    if x21 < y21:
        c = False
    if x21 == y21:
        if x20 < y20:
            c = False
    if c:
        a += 1
        y += 1
        x1 -= func1()
        x0 -= (y << i) % MOD  # 下 16 ビットをとる
        if x0 < 0:
            x1 -= 1
            x0 += MOD  # 下 16 ビットをとる
        y += 1

print(hex(a), '=', a / 256)
```

図 2. 平方根を計算するプログラム（Python）

6.2 テストプログラム（機械語，アセンブリ）

Python プログラムが正しく動作したので，アセンブリプログラムを書き，さらに機械語に書き直して実装した．以下の図 3 に機械語とアセンブリのプログラムを示す．

機械語	アセンブリ			
3000		LAD	GR0	#00
3101		LAD	GR1	#01
3210		LAD	GR2	#10
2300		LD2	GR3	#00
1400		LD1	GR4	GR0
1500		LD1	GR5	GR0
1600		LD1	GR6	GR0
1700		LD1	GR7	GR0
1800		LD1	GR8	GR0
1900		LD1	GR9	GR0
1a00		LD1	GR10	GR0
1b00		LD1	GR11	GR0
1c00		LD1	GR12	GR0
1d03		LD1	GR13	GR3
8d01	LOOP0	SR	GR13	GR1
b092		JZE		QUIT0
5701		ADD	GR7	GR1
a08e		JMP		LOOP0
5702	QUIT0	ADD	GR7	GR2
1d07		LD1	GR13	GR7
9d01		NAND	GR13	GR1
9d0d		NAND	GR13	GR13
570d		ADD	GR7	GR13
6700	LOOP1	SUB	GR7	GR0
c0b7		JMI		QUIT1
1807		LD1	GR8	GR7
afee		JMP	GR15	USE16
6802		SUB	GR8	GR2
7501		SL	GR5	GR1
7601		SL	GR6	GR1

afda	GETY21	JMP	GR15	_Y21
afdf	GETY20	JMP	GR15	_Y20
afc6	GOFUNC0	JMP	GR15	FUNC0
afe5	GETX21	JMP	GR15	_X21
afe8	GETX20	JMP	GR15	_X20
690b	IF0	SUB	GR9	GR11
d0b4		JOV		ENDIF0
b0a7	ELIF0	JZE		IF00
a0a9		JMP		ELSE0
6a0c	IF00	SUB	GR10	GR12
d0b4		JOV		ENDIF0
5501	ELSE0	ADD	GR5	GR1
5601		ADD	GR6	GR1
afd0	GOFUNC1	JMP	GR15	FUNC1
630d		SUB	GR3	GR13
1d06		LD1	GR13	GR6
7d07		SL	GR13	GR7
640d		SUB	GR4	GR13
d0b2	IF1	JOV		_PROC1
a0b3		JMP		ENDIF1
6301	_PROC1	SUB	GR3	GR1
5601	ENDIF1	ADD	GR6	GR1
afec	ENDIF0	JMP	GR15	USE2
6702		SUB	GR7	GR2
a097		JMP		LOOP1
1005	QUIT1	LD1	GR0	GR5
3404		LAD	GR4	#04
3808		LAD	GR8	#08
3c0c		LAD	GR12	#0C
1100		LD1	GR1	GR0
1200		LD1	GR2	GR0
1300		LD1	GR3	GR0
8104		SR	GR1	GR4
8208		SR	GR2	GR8
830c		SR	GR3	GR12
f000		DISP	GR0	GR0

f101		DISP	GR1	GR1
f202		DISP	GR2	GR2
f303		DISP	GR3	GR3
0000		HALT	680	GR0
6800	FUNC0	SUB	GR8	_MI0
c0cb		JMI		GR3
1d03	_PL0	LD1	GR13	GR8
8d08		SR	GR13	#00
ef00		RJMP	GR15	GR8
3e00	_MI0	LAD	GR14	GR3
6e08		SUB	GR14	GR14
1d03		LD1	GR13	GR0
7d0e		SL	GR13	_MI1
ef00		RJMP	GR15	GR6
6800	FUNC1	SUB	GR8	GR8
c0d5		JMI		#00
1d06	_PL1	LD1	GR13	GR8
7d08		SL	GR13	GR6
ef00		RJMP	GR15	GR14
3e00	_MI1	LAD	GR14	_OV1
6e08		SUB	GR14	GR0
1d06		LD1	GR13	GR1
8d0e		SR	GR13	GR6
ef00		RJMP	GR15	GR1
d0dd	_Y21	JOV		GR12
1b00	_NOV1	LD1	GR11	GR13
ef00		RJMP	GR15	GR13
1b01	_OV1	LD1	GR11	GR3
ef00		RJMP	GR15	GR7
1c06	_Y20	LD1	GR12	GR4
1d01		LD1	GR13	GR7
9c0c		NAND	GR12	GR13
9d0d		NAND	GR13	#02
9c0d		NAND	GR12	#10
ef00		RJMP	GR15	#00
1903	_X21	LD1	GR9	#01

8907		SR	GR9	#10
ef00		RJMP	GR15	#00
1a04	_X20	LD1	GR10	GR0
8a07		SR	GR10	GR0
5a0d		ADD	GR10	GR0
ef00		RJMP	GR15	GR0
3202	USE2	LAD	GR2	GR0
ef00		RJMP	GR15	GR0
3210	USE16	LAD	GR2	GR0
ef00		RJMP	GR15	GR0

図 3. 平方根を計算するプログラム（機械語，アセンブリ）

6.3 シミュレーション

シミュレーション結果を抜粋する．以下の図 4 では，入力値が gr3，出力値が gr5 に現れる．メモリの 0 番地に c350 を置き，$\sqrt{50000} = 223.60\ldots$（c350 → df9b）を計算させたところ，意図通りの結果が現れている．

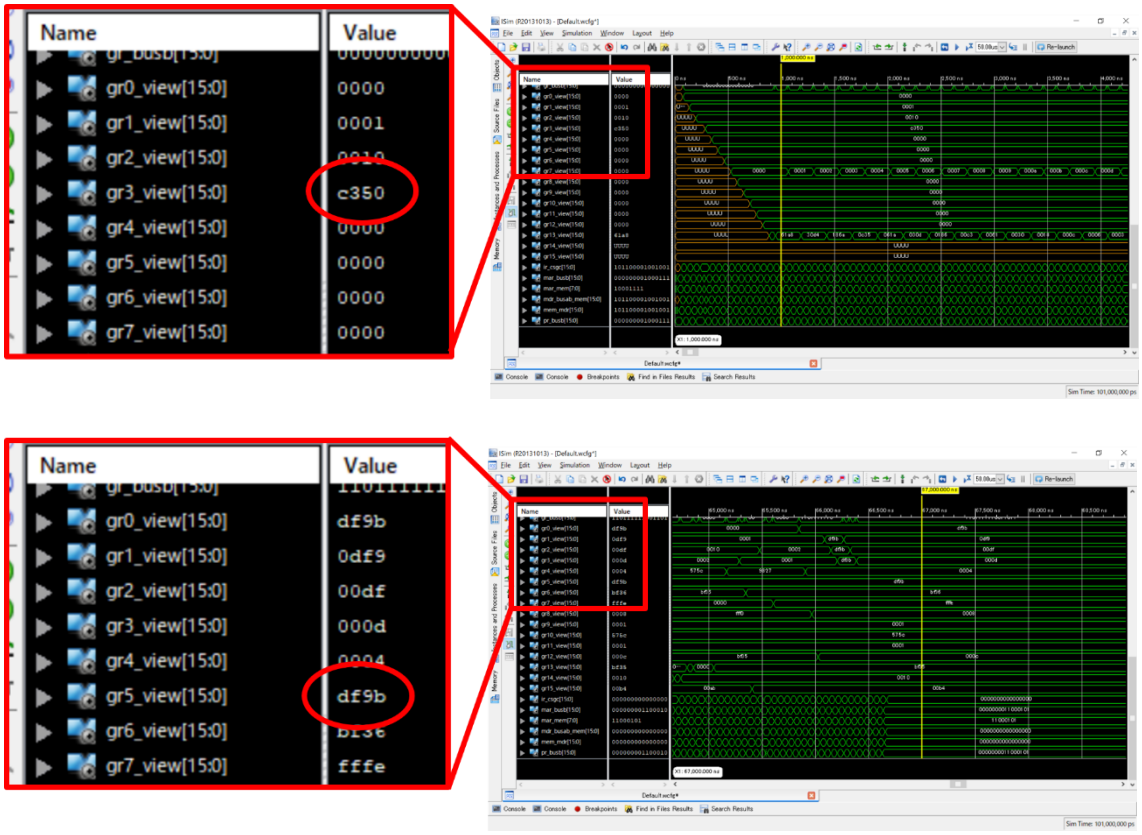

図 4. シミュレーション結果（上：1ns 時点，下：67ns 時点）

7 FPGA 実装

クロック信号とメモリは基板上に存在するため，それらを利用するようにプログラムを書き直す．メモリ利用プログラムは自動生成したものを修正した．

また入出力インターフェースを利用するために，マッピングという設定を行った．同時に 7 セグメント LED と 0〜f の値を対応させるプログラムを書いた．

7.1 動作例

ボタン 0 を押しながらスイッチを操作すると，下 8bit が入力できる．同様にボタン 1 を押しながら操作すると上 8bit が入力できる．入力が完了したときボタン 2 を押すと，計算結果が表示される仕組みとなっている．

シミュレーションと同じく $\sqrt{50000} = 223.60$ …(c350 → df.9b)の計算を行ったところ，以下の図 5 のように正しく動作した．

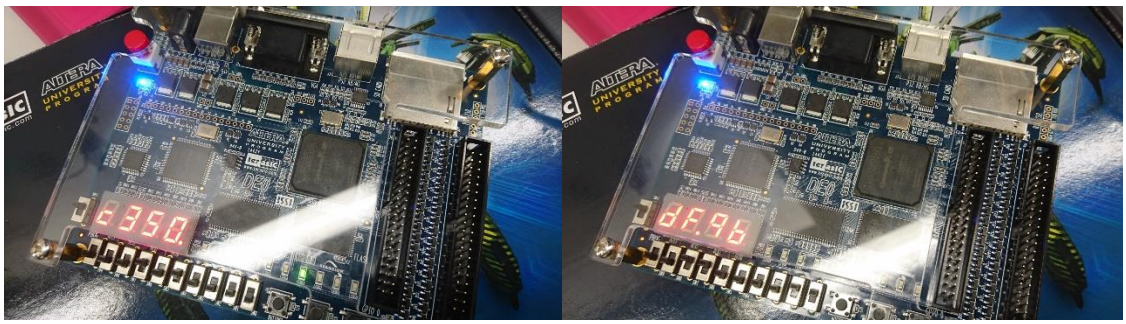

図 5．FPGA 動作例

8 おわりに

当初の予定通り，16 進数で 0000〜ffff の値の平方根を小数第 2 位まで求めるプログラムを作成することができた．FPGA へ実装することもできたので，目標はほぼ達成した．

改良点を挙げるとすれば，入出力をコンポーネントとして実装し直したり，小数計算を平方根計算とは別にアルゴリズム化したりしたい．

実験を通し，VHDL の基本的な考え方や仕様を理解することができた．