ALTERA FPGA への接続 (追加資料)

須山 敬之
NTT コミュニケーション科学研究所
suyama@cslab.kecl.ntt.co.jp

1 はじめに

本稿は講習会資料に間に合わなかった分の追加資料です. ここでは対象とするデバイスとして FLEX10K シリーズを用います. デバイスの詳細については [1], [2] から得られる情報を参照してください. また, 題材として 8 ビット CPU (KUE-CHIP2) を取り上げています. KUE-CHIP2 については [3] の第 4 回 PARTHENON 講習会のページを参照してください. この文書の内容については正確さを損なわないように努力していますが, 経験的に得た情報も含まれているため, より良い方法がある可能性があることをご了承下さい.

2 MAX+plus II におけるマッピングの向上

FLEX10K のロジック・エレメントはキャリー・イン, キャリー・アウト, カスケード・イン, カスケード・アウトといった上下に配置されている他のロジック・エレメントと直接接続されている特別の結線を持っています. これらはアダーや多入力のロジックを実現するために有効ですが, デフォルトのコンパイラーのセッティングでは使用されません. これらの結線をコンパイル時に使用する設定は以下の通りです.

1. `Compiler` を起動します.
2. `Assign→Global Project Logic Synthesis...` を開きます.
3. `Global Project Synthesis Style` を `NORMAL` から `FAST` に変更し, `OK` を押します.

`Global Project Synthesis Style` 内の `Define Synthesis Style...` で, `NORMAL` から `FAST` に変更する際に, `Carry Chain` 及び `Cascade Chain` の設定がそれぞれ `Ignore` から `Auto` になることが確認できます.

3 ALTERA のライブラリの利用

MAX+plus II ではあらかじめ様々な機能のマクロをライブラリとして用意しています. どのようなマクロが用意されているかは MAX+plus II に付属のヘルプファイルを参照してください.

ライブラリを使う利点として以下のような点が考えられます.

- ライブラリに含まれるマクロは FPGA メーカが FPGA の構造を考慮して作成されていると考えられるため, 面積, 速度的に有利である可能性がある.
- 一般的に良く使われる機能がマクロとして作成されているため, 既に存在する機能を作成する手間が省ける.

また, 欠点として以下のような点が考えられます.

- SFL 記述が特定のメーカに依存した記述になり, 可搬性が損なわれる.
- `seconds` によるシミュレーションができなくなる. これはそのマクロと等価な記述を用意することにより可能になります. また, そのための記述は `circuit` 文でよいため, 算術演算などやや柔軟な記述ができます.

この欠点を解消するために ALTERA 社などは LPM (Library of Parameterized Module) を提唱していますが, 今のところそれほど一般的ではありません.

ここでは ALU と組み込みメモリを例にあげ, ライブラリの利用法について説明致します.

3.1 KUE-CHIP2

ここではKUE-CHIP2を例題として取り上げます. 図1はKUE-CHIP2のブロック図です. ここでのKUE-CHIP2は[4]を元にして記述されています[1].

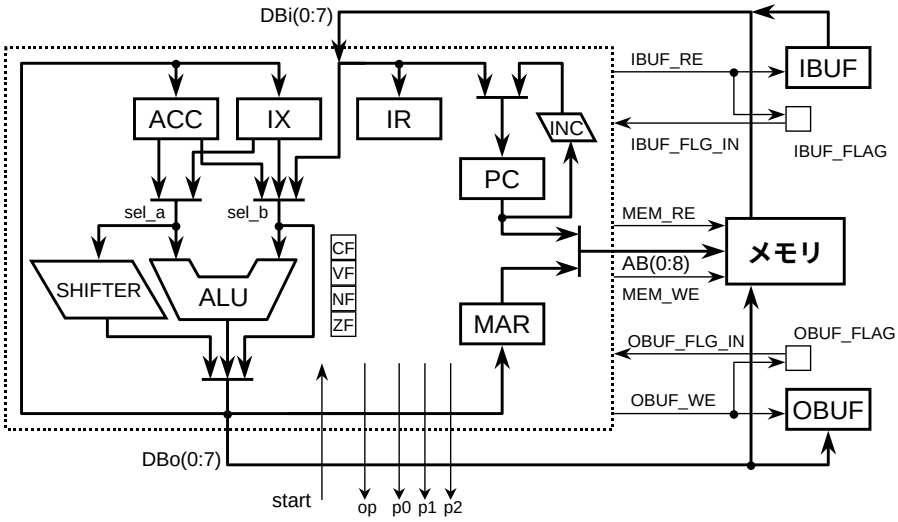

図 1: KUE-CHIP2 のブロック図

3.2 ALU

図1中にあるALUは表1の動作をするよう規定されています. すなわち8ビットの入力A, B及び1ビットのキャリーに対する演算をmmmmで制御するようになっています. 通常, SFL記述のみでこれらの全ての機能を実現する場合には, まず更に下位のモジュールとして8ビットのadderを用意し, 加算 / 減算についてはmmmmの値に応じ, adderに入力値をそのまま入れるかまたは反転入力し, それ以外の演算はそれぞれ独自に実現します. リスト1にALUのSFL記述を示します.

表 1: ALU の動作

略記号	命令	mmm	命令の内容	CF	VF	NF	ZF
SBC	SuB with Carry	000	A := A + ^B + ^CF	^C	V	N	Z
ADC	ADd with Carry	001	A := A + B + CF	C	V	N	Z
SUB	SUBtraction	010	A := A + ^B + 0b1	-	V	N	Z
ADD	ADDition	011	A := A + B	-	V	N	Z
EOR	Exclusive OR	100	A := A @ B	-	0	N	Z
OR	OR	101	A := A | B	-	0	N	Z
AND	AND	110	A := A & B	-	0	N	Z
CMP	CoMPare	111	A + ^B + 0b1	-	V	N	Z

C: MSB(最上位ビット)からの桁上げが起こったときに1となり、その他の場合は0となる
V: MSBからの桁上げとMSBへの桁上げのどちらか1つのみ起こったときに1となり、その他の場合は0となる
N: 結果においてMSBが1であれば1となり、その他の場合は0となる
Z: 結果においてすべてのビットが0であれば1となり、その他の場合は0となる

リスト 1: SFL による ALU の記述

```
1  /** kuealu: ALU for KUE-CHIP2 **/
```

[1]同じ情報は[3]にもあります.

```
 2
 3  module add8 {
 4      /** external pins **/
 5      instrin     do;
 6      input       a<8>, b<8>, ci;
 7      output      out<8>, co, vo;
 8
 9      /** elements **/
10      sel         c0, c1, c2, c3, c4, c5, c6, c7;
11
12      /** operations of instrin **/
13      instruct do par {
14          c0 = (a<0> & b<0>) | (b<0> & ci) | (ci & a<0>);
15          c1 = (a<1> & b<1>) | (b<1> & c0) | (c0 & a<1>);
16          c2 = (a<2> & b<2>) | (b<2> & c1) | (c1 & a<2>);
17          c3 = (a<3> & b<3>) | (b<3> & c2) | (c2 & a<3>);
18          c4 = (a<4> & b<4>) | (b<4> & c3) | (c3 & a<4>);
19          c5 = (a<5> & b<5>) | (b<5> & c4) | (c4 & a<5>);
20          c6 = (a<6> & b<6>) | (b<6> & c5) | (c5 & a<6>);
21          c7 = (a<7> & b<7>) | (b<7> & c6) | (c6 & a<7>);
22          out = (a<7> @ b<7> @ c6) ||
23                (a<6> @ b<6> @ c5) ||
24                (a<5> @ b<5> @ c4) ||
25                (a<4> @ b<4> @ c3) ||
26                (a<3> @ b<3> @ c2) ||
27                (a<2> @ b<2> @ c1) ||
28                (a<1> @ b<1> @ c0) ||
29                (a<0> @ b<0> @ ci);
30          co = c7;
31          vo = c7 @ c6;
32      }
33  }
34
35  declare add8 {
36      /** external pins **/
37      instrin     do;
38      input       a<8>, b<8>, ci;
39      output      out<8>, co, vo;
40      /** arguments of instrin **/
41      instr_arg   do(a, b, ci);
42  }
43
44  module kuealu {
45      /** external pins **/
46      instrin     do;
47      input       mode<3>, a<8>, b<8>, ci;
48      output      out<8>, co, vo, no, zo;
49
50      /** elements **/
51      add8        add;
52
53      /** operations of instrin **/
54      instruct do par {
55          any { /* out */
56              mode == 0b000 : out = add.do(a, ^b, ^ci).out; /* SBC */
57              mode == 0b001 : out = add.do(a, b, ci).out;   /* ADC */
58              mode == 0b010 : out = add.do(a, ^b, 0b1).out; /* SUB */
59              mode == 0b011 : out = add.do(a, b, 0b0).out;  /* ADD */
60              mode == 0b100 : out = a @ b;                  /* EOR */
61              mode == 0b101 : out = a | b;                  /* OR  */
62              mode == 0b110 : out = a & b;                  /* AND */
63              mode == 0b111 : out = add.do(a, ^b, 0b1).out; /* CMP */
64          }
65          any { /* co */
66              mode == 0b000 : co = ^add.co; /* SBC */
67              mode == 0b001 : co = add.co;  /* ADC */
68              else : co = ci; /* SUB, ADD, EOR, OR, AND, CMP */
69          }
70          any { /* vo */
71              mode == 0b100 : vo = 0b0; /* EOR */
72              mode == 0b101 : vo = 0b0; /* OR  */
73              mode == 0b110 : vo = 0b0; /* AND */
74              else : vo = add.vo; /* SBC, ADC, SUB, ADD, CMP */
75          }
76          no = out<7>;
77          zo = ^(/| out);
78      }
```

表 2: KUE-CHIP2 の ALU と ALTERA のマクロの比較

制御信号	KUE-CHIP2		ALTERA (74382)	
000	SuB with Carry	A + ^B + ^CF	Clear	0
001	ADd with Carry	A + B + CF	B − A	B − A − Cn
010	SUBtraction	A + ^B + 0b1	A − B	A − B − Cn
011	ADDition	A + B	A + B	A + B + Cn
100	Exclusive OR	A @ B	A $ B	A $ B
101	OR	A | B	A # B	A # B
110	AND	A & B	A & B	A & B
111	CoMPare	A + ^B + 0b1	Preset	1

```
79  }
```

ALTERA では, 数種類の ALU のマクロを用意しています. ここではその中で 74382 の ALU を用います (図 2). これは入力が 4 ビットの ALU で, 2 個の 74382 をカスケードに繋ぐことにより, 8 ビットの ALU を実現することができます.

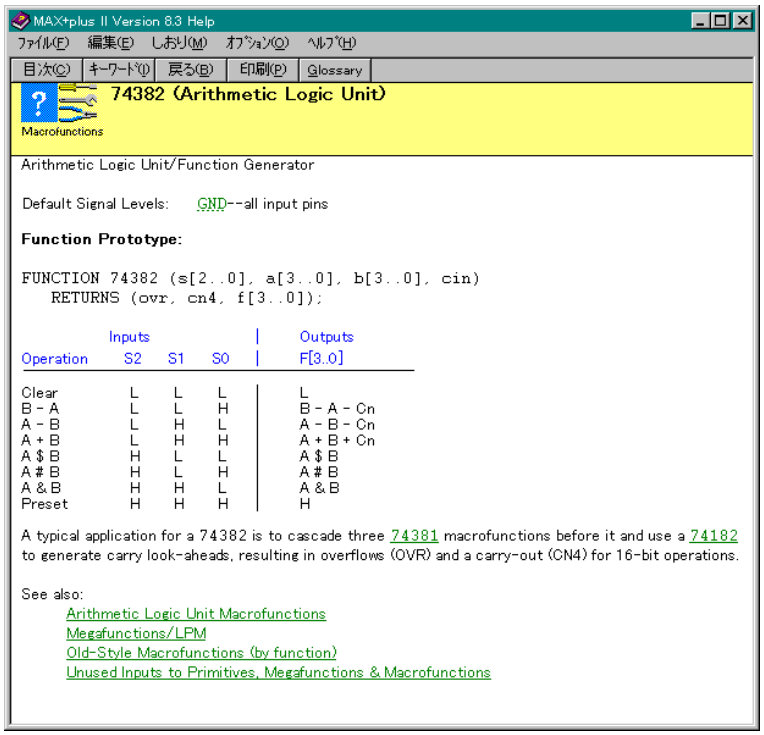

図 2: ALTERA で用意されている ALU

KUE-CHIP2 の ALU と ALTERA で用意されている ALU を比較するとかなり似た仕様ですが, 一部異なっている部分もあります. 本来は ALTERA で用意されている ALU に添った設計を行えば良いのですが, 今回は KUE-CHIP2 の仕様があらかじめ決まっているので, その差異を吸収するように SFL の記述を行います. 初めに, その違いを見ていきます. 動作を制御する制御信号 (mmm) はどちらも 3 ビットで共通です. それでは制御信号ごとに割り当てられている動作を見ます.

表 2に KUE-CHIP2 の ALU と ALTERA で用意されている ALU の仕様の比較を示します. 制御信号が 000, 001, 111 の場合, 両者は異なった動作をしますが, それ以外の場合は同じ動作をします[2]. リスト 2 はこれらの差異を吸収

[2] 100, 101 では記号は異なっていますが, 同じ意味です.

するようにしたSFL記述です.

リスト 2: 74382 のマクロを用いた ALU の記述

```
 1  declare alu382 {
 2      input       a<4>, b<4>, s<3>, cin;
 3      output      f<4>, ovr, cn4;
 4      instrin     do;
 5      instr_arg   do(a, b, s, cin);
 6  }
 7
 8  module alu8 {
 9      input       a<8>, b<8>, s<3>, cin;
10      output      f<8>, ovr, cn8;
11      alu382    alu4_1, alu4_2;
12      instrin     do;
13
14      sel_v       cn4;
15
16      instruct do par {
17          f = alu4_2.do(a<7:4>, b<7:4>, s, cn4).f
18           || alu4_1.do(a<3:0>, b<3:0>, s, cin).f;
19          cn4 = alu4_1.cn4;
20          ovr = alu4_2.ovr;
21          cn8 = alu4_2.cn4;
22      }
23  }
24
25  declare alu8 {
26      input       a<8>, b<8>, s<3>, cin;
27      output      f<8>, ovr, cn8;
28      instrin     do;
29      instr_arg   do(a, b, s, cin);
30  }
31
32  module kuealu {
33      /** external pins **/
34      instrin     do;
35      input       mode<3>, a<8>, b<8>, ci;
36      output      out<8>, co, vo, no, zo;
37
38      /** elements **/
39      alu8        alu;
40
41      /** operations of instrin **/
42      instruct do par {
43          any { /* out */
44              mode == 0b000 : out = alu.do(a, b, 0b101,  ^ci).f; /* SBC */
45              mode == 0b001 : out = alu.do(a, b, 0b011,  ci).f; /* ADC */
46              mode == 0b010 : out = alu.do(a, b, 0b010, 0b1).f; /* SUB */
47              mode == 0b111 : out = alu.do(a, b, 0b010, 0b1).f; /* CMP */
48              else :          out = alu.do(a, b,  mode, 0b0).f; /* ADD, EOR, OR, AND */
49          }
50          any { /* co */
51              mode == 0b000 : co = alu.cn8;  /* SBC */
52              mode == 0b001 : co = alu.cn8;  /* ADC */
53              mode == 0b010 : co = ^alu.cn8;  /* ADC */
54              else : co = ci; /* SUB, ADD, EOR, OR, AND, CMP */
55          }
56          any { /* vo */
57              mode == 0b100 : vo = 0b0; /* EOR */
58              mode == 0b101 : vo = 0b0; /* OR  */
59              mode == 0b110 : vo = 0b0; /* AND */
60              else : vo = alu.ovr; /* SBC, ADC, SUB, ADD, CMP */
61          }
62          no = out<7>;
63          zo = ^(/| out);
64      }
65  }
```

alu382 のインターフェースの宣言部 (1 行目から 6 行目)

ここではALTERAのマクロである74382のインターフェースの定義を行っています. SFLで記述する際のモジュール名の一文字目は英文字でなければなりませんので, alu382という名前とします. ALTERA側との名前の整合はラ

イブラリマッピングファイルで取ります．入出力端子名は ALTERA での定義と合わせています．その他に制御入力端子 (do) が加えられています．この端子は SFL 記述でモジュールを起動する際に必要となりますが, 回路を合成する時に削除されます (詳しくは後述)．seconds でシミュレーションを行うためには alu382 の機能を記述した SFL が必要となりますが, ここではその記述は行いません．

モジュール alu8 の定義部 (8 行目から 23 行目)

KUE-CHIP2 で用いられる ALU は 8bit ですので，4bit の ALU である上述の alu382 を 2 つ用いて 8bit の ALU を作成します．

alu8 のインターフェースの宣言部 (25 行目から 30 行目)

ここでは alu8 のインターフェースを宣言しています．

モジュール kuealu の定義部 (32 行目から 65 行目)

ここでは kuealu の定義をしています．外部入出力端子は元々の設計と互換性を保つため, リスト 1 と同一の定義とします． 42 行目以降で動作の定義を行っています． 表 2で見たように, ALU の制御端子の意味が異なっている部分がありますので, その部分は個別に定義し直し, それ以外の部分はそのまま ALU への入力とします．制御信号が 000, 001, 111 の場合の動作が異なっていると前述しましたが, 他にも実際には減算の場合のキャリーの極性が異なっているという違いがありますので, 減算の時も個別に定義し直しています．

3.2.1 PARTHENON による論理合成

PARTHENON で論理合成を行うためには，全ての最下位構成要素が PCD として用意されている必要があります．今回は alu382 を最下位要素として使用しましたので, それに対応する PCD を新たに記述し, ライブラリに追加します．新たに作成した PCD ファイルは $PARTHENON/cell_lib.dir/ALTERA/altera/ 等の下に start_ex.dir, cell_ex.dir がありますので, その中に置くと良いでしょう．これらのディレクトリは初期状態では何も置かれていませんが, auto コマンドを用いた場合はライブラリのサーチパスに既に組み込まれていますので, 新たに作成した PCD ファイルが読み込まれます．

リスト 3 に alu382.pcd を示します．リスト 3 中の power, area 等の値は根拠のある値ではありません．入力ピン (元々は制御入力端子) の do(14 行目) は, type を unused として定義します．この様にするとこのピンは最適化中に削除されます[3]．また, m_clock, s_clock, b_clock, p_reset(15 行目から 18 行目) は論理合成時に一旦自動的に追加されますので, PCD で宣言する必要がありますが, これらも unused と定義することで, 削除されます[4]．

リスト 3: alu382.pcd

```
 1  (def-module alu382 power 100 area 10 gates 10
 2          (def-pin a[0] type input load 0.1)
 3          (def-pin a[1] type input load 0.1)
 4          (def-pin a[2] type input load 0.1)
 5          (def-pin a[3] type input load 0.1)
 6          (def-pin b[0] type input load 0.1)
 7          (def-pin b[1] type input load 0.1)
 8          (def-pin b[2] type input load 0.1)
 9          (def-pin b[3] type input load 0.1)
10          (def-pin s[0] type input load 0.1)
11          (def-pin s[1] type input load 0.1)
12          (def-pin s[2] type input load 0.1)
13          (def-pin cin type input load 0.1)
14          (def-pin do type unused)
15          (def-pin m_clock type unused)
16          (def-pin s_clock type unused)
17          (def-pin b_clock type unused)
18          (def-pin p_reset type unused)
19          (def-pin f[0] type output load 0.1)
20          (def-pin f[1] type output load 0.1)
21          (def-pin f[2] type output load 0.1)
22          (def-pin f[3] type output load 0.1)
```

[3] もちろん, 実際に使用する場合はこの様にしてはいけません．

[4] これは ALU が組合わせ回路であるためです．

```
23          (def-pin cn4 type output load 0.1)
24          (def-pin ovr type output load 0.1)
25
26          (def-delay /a[0] /f[0]  (+ 1.1 (* 3.0 ~f[0])))
27          (def-delay /a[0] /f[1]  (+ 1.1 (* 3.0 ~f[1])))
28          (def-delay /a[0] /f[2]  (+ 1.1 (* 3.0 ~f[2])))
29          (def-delay /a[0] /f[3]  (+ 1.1 (* 3.0 ~f[3])))
30          (def-delay /a[0] /cn4  (+ 1.1 (* 3.0 ~cn4)))
31          (def-delay /a[0] /ovr  (+ 1.1 (* 3.0 ~ovr)))
32  )
```

以上の準備で PARTHENON での論理合成が可能となります.

3.2.2 MAX+plus Ⅱでのマッピング

[5] で述べたように, PARTHENON から出力された EDIF の内容と MAX+plus Ⅱが理解できる EDIF の対応を取るためにはライブラリマッピングファイルを用います. MAX+plus Ⅱではライブラリマッピングファイルを二つまで指定できるので, partenon.lmf とは別に新たに par2.lmf (リスト 3) を作成し, MAX+plus Ⅱで指定します.

リスト 3: par2.lmf

```
1  LIBRARY par2
2
3  % alu(74382) %
4  BEGIN
5  FUNCTION 74382 ( CIN, B0, A0, B1, A1, B2, A2, B3, A3, S0, S1, S2 )
6  RETURNS  ( F0, F1, F2, F3, CN4, OVR )
7  FUNCTION "alu382"("cin","b_0_","a_0_","b_1_","a_1_","b_2_","a_2_","b_3_","a_3_", "s_0_","s_1_","s_2_")
8  RETURNS  ("f_0_","f_1_","f_2_","f_3_","cn4","ovr")
9  END
```

3.3 組み込みメモリ

ALTERA FLEX10K シリーズ等ではその内部にメモリとして使用可能な EAB (Embedded Array Block) と呼ばれる部分を持っています[5]. それを利用することにより, 図 1に示される全てを一つのチップの中に納めることができます.

ここでは [4] の board.sfl を元に全体を記述します. board.sfl は seconds でのシミュレーション用に記述された SFL 合成用ではありません.

リスト 4: オリジナルの board.sfl

```
1  circuit r512_8 {
2      /** external pins **/
3      instrin     read, write;
4      input       adrs<9>, din<8>;
5      output      dout<8>;
6      /** elements **/
7      mem         cell[512]<8>;
8      /** operations of instrin **/
9      instruct    read dout = cell[adrs];
10     instruct    write cell[adrs] := din;
11 }
12
13 declare kuechip2 {
14     /** external pins **/
15     instrin     start;
16     input       dbi<8>;
17     input       ibuf_flg_in, obuf_flg_in;
18     instrout    mem_we, mem_re;
19     instrout    ibuf_re, obuf_we;
20     output      dbo<8>, ab<9>;
21 }
22
23 declare r512_8 {
24     /** external pins **/
25     instrin     read, write;
```

[5] 詳細は [1], [2] を参照

```
26      input       adrs<9>, din<8>;
27      output      dout<8>;
28      /** arguments of instrin **/
29      instr_arg   read(adrs);
30      instr_arg   write(adrs, din);
31  }
32
33  module board {
34      /** elements **/
35      reg_wr      ibuf<8>, ibuf_flg, obuf<8>, obuf_flg;
36      kuechip2    cpu;
37      r512_8      mem;
38
39      /** operations in common **/
40      par {
41          cpu.ibuf_flg_in = ibuf_flg;
42          cpu.obuf_flg_in = obuf_flg;
43      }
44
45      /** operations of instrout **/
46      instruct cpu.mem_we mem.write(cpu.ab, cpu.dbo);
47      instruct cpu.mem_re cpu.dbi = mem.read(cpu.ab).dout;
48      instruct cpu.ibuf_re par {
49          cpu.dbi = ibuf;
50          ibuf_flg := 0b0;
51      }
52      instruct cpu.obuf_we par {
53          obuf := cpu.dbo;
54          obuf_flg := 0b1;
55      }
56  }
```

リスト 5: 新たに記述を加えた board.sfl

```
 1  circuit r512_8 {
 2      /** external pins **/
 3      instrin     read, write;
 4      input       adrs<9>, din<8>;
 5      output      dout<8>;
 6      /** elements **/
 7      mem         cell[512]<8>;
 8      /** operations of instrin **/
 9      instruct    read dout = cell[adrs];
10      instruct    write cell[adrs] := din;
11  }
12
13  declare kuechip2 {
14      /** external pins **/
15      instrin     start;
16      input       dbi<8>;
17      input       ibuf_flg_in, obuf_flg_in;
18      instrout    mem_we, mem_re;
19      instrout    ibuf_re, obuf_we;
20      instrout    op, p0, p1, p2;
21      output      dbo<8>, ab<9>;
22
23      output      ir_out<8>, pc_out<8>, acc_out<8>, ix_out<8>;
24  }
25
26  declare r512_8 {
27      /** external pins **/
28      instrin     read, write;
29      input       adrs<9>, din<8>;
30      output      dout<8>;
31      /** arguments of instrin **/
32      instr_arg   read(adrs);
33      instr_arg   write(adrs, din);
34  }
35
36  %i "kuechip2.sfl"
37
38  module board {
39      /** elements **/
40      reg_wr      ibuf<8>, ibuf_flg, obuf<8>, obuf_flg;
41      kuechip2    cpu;
42      r512_8      mem;
```

```
43      input       ibuf_in<8>;
44      output      obuf_out<8>;
45      output      ibuf_flg_out, obuf_flg_out;
46      instrin     inst_ibuf_in, inst_obuf_out;
47
48      input       mem_addr<9>, mem_w_data<8>;
49      output      mem_r_data<8>;
50      instrin     mem_r, mem_w;
51
52      output      cpu_op, cpu_p0, cpu_p1, cpu_p2;
53      output      cpu_ab<9>, cpu_dbo<8>;
54      instrin     cpu_start;
55
56      output      ir_out<8>, pc_out<8>, acc_out<8>, ix_out<8>;
57
58      /** operations in common **/
59      par {
60          cpu.ibuf_flg_in = ibuf_flg;
61          cpu.obuf_flg_in = obuf_flg;
62          ibuf_flg_out = ibuf_flg;
63          obuf_flg_out = obuf_flg;
64          cpu_op = cpu.op;
65          cpu_p0 = cpu.p0;
66          cpu_p1 = cpu.p1;
67          cpu_p2 = cpu.p2;
68          cpu_ab = cpu.ab;
69          cpu_dbo = cpu.dbo;
70          ir_out = cpu.ir_out;
71          pc_out = cpu.pc_out;
72          acc_out = cpu.acc_out;
73          ix_out = cpu.ix_out;
74      }
75
76      instruct inst_ibuf_in par {
77          ibuf := ibuf_in;
78          ibuf_flg := 0b1;
79      }
80
81      instruct inst_obuf_out par {
82          obuf_out = obuf;
83          obuf_flg := 0b0;
84      }
85
86      instruct mem_r par {
87          mem_r_data = mem.read(mem_addr).dout;
88      }
89
90      instruct mem_w par {
91          mem.write(mem_addr, mem_w_data);
92      }
93
94      instruct cpu_start par {
95          cpu.start();
96      }
97
98      /** operations of instrout **/
99      instruct cpu.mem_we mem.write(cpu.ab, cpu.dbo);
100     instruct cpu.mem_re cpu.dbi = mem.read(cpu.ab).dout;
101     instruct cpu.ibuf_re par {
102         cpu.dbi = ibuf;
103         ibuf_flg := 0b0;
104     }
105     instruct cpu.obuf_we par {
106         obuf := cpu.dbo;
107         obuf_flg := 0b1;
108     }
109 }
```

元のままでは KUE-CHIP2 の本体がないため合成できませんので, ここでは本体の SFL 記述をインクルード (リスト 5:36 行目) することにより合成を可能にしています. その方法以外にも別々に合成し, 後に統合する方法もありますが, ここでは説明致しません.

新たな外部端子の定義 (43 行目から 54 行目)

オリジナルの board.sfl は外部への入出力端子が定義されていません. このまま論理合成すると全ての回路が不要なものと判断され, 回路が全くなくなってしまいますし, また実際に何も制御や観測できませんので, そのための端子を付け加えます.

共通動作の定義 (59 行目から 74 行目)

ここでは常に動いている動作を記述しています. 具体的には CPU からの出力を更に外部への出力端子に出力する等の動作です.

バッファ, メモリへの入出力動作の定義 (76 行目から 92 行目)

外部から, obuf , ibuf 及び mem に対して読み書きを行う動作の定義を行っています.

CPU を起動するための動作の定義 (76 行目から 92 行目)

ここでは CPU を起動するための定義を行っています.

3.3.1 PARTHENON による論理合成

PARTHENON で論理合成するためには ALU の場合と同様に, メモリを PCD として用意する必要があります. KUE-CHIP2 ではアドレスバスが 9bit, データバスが 8bit のメモリを使用します. 一方, ALTERA でのメモリは LPM として定義されています. LPM ではデータバス幅, アドレスバス幅などがパラメータ化されていて, 様々な大きさのメモリに柔軟に対応できるようにしています. また, ALTERA ではデータのインプットとアウトプットのポートが同一のマクロ (lpm_ram_io) と別となっているマクロ (lpm_ram_dq) の 2 種類が用意されておりますが, ここでは KUE-CHIP2 での定義と同じ入出力のポートが別となっているマクロを用います. これらの詳細については [6] に記述されていますので, そちらを参照してください.

表 3: lpm_ram_dq の入出力端子

Name	Direction	Required	Description
data[ ]	input	yes	data input
address[ ]	input	yes	address input
we	input	yes	write enable
inclock	input	no	synchronizes memory loading
outclock	input	no	synchronizes q outputs
q[ ]	output	yes	data output

表 4: lpm_ram_dq のパラメータ (一部)

Name	Type	Required	Description
lpm_width	integer	yes	width of data[ ] and q[ ]
lpm_widthad	integer	yes	width of the address port
lpm_indata	string	no	controls whether the data port is registered
lpm_outdata	string	no	controls whether the q port is registered
lpm_address_control	string	no	controls whether the address and we ports are registered
lpm_file	string	no	name of memory initialization file

表 3, 4に lpm_ram_dq の仕様の一部を示します．inclock, outclock はそれらを使用することにより, メモリの動作をクロックと同期して使用することができます．それらを使用しない場合は非同期動作をします．ここでは inclock, outclock を両方使用し, 同期動作をさせます.

上記のように KUE-CHIP2 で想定しているメモリと実際に ALTERA で用意されているメモリでは仕様が異なっています. その差異は PARTHENON のライブラリで吸収します. まず初めに, ALTERA の lpm_ram_dq に対応する PCD を記述します (リスト 6).

リスト 6: lpm_ram_dq に対応する PCD (lpm_ram.pcd)

```
 1  (def-module lpm_ram power 13106 area 6553 gates 6553
 2  (def-pin adrs[0] type input load 0.05)
 3  (def-pin adrs[1] type input load 0.05)
 4  (def-pin adrs[2] type input load 0.05)
 5  (def-pin adrs[3] type input load 0.05)
 6  (def-pin adrs[4] type input load 0.05)
 7  (def-pin adrs[5] type input load 0.05)
 8  (def-pin adrs[6] type input load 0.05)
 9  (def-pin adrs[7] type input load 0.05)
10  (def-pin adrs[8] type input load 0.05)
11  (def-pin din[0] type input load 0.05)
12  (def-pin din[1] type input load 0.05)
13  (def-pin din[2] type input load 0.05)
14  (def-pin din[3] type input load 0.05)
15  (def-pin din[4] type input load 0.05)
16  (def-pin din[5] type input load 0.05)
17  (def-pin din[6] type input load 0.05)
18  (def-pin din[7] type input load 0.05)
19  (def-pin dout[0] type output load 0.05)
20  (def-pin dout[1] type output load 0.05)
21  (def-pin dout[2] type output load 0.05)
22  (def-pin dout[3] type output load 0.05)
23  (def-pin dout[4] type output load 0.05)
24  (def-pin dout[5] type output load 0.05)
25  (def-pin dout[6] type output load 0.05)
26  (def-pin dout[7] type output load 0.05)
27  (def-pin we type input load 0.05)
28  (def-pin in_clock type input load 0.05 note clock)
29  (def-pin out_clock type input load 0.05 note clock)
30  (def-pin s_clock type unused)
31  (def-pin m_clock type unused)
32  (def-pin p_reset type unused)
33  (def-delay /adrs[0] /dout[0] (+ 10 (* 5 ~dout[0])))
34  (def-delay /adrs[0] /dout[1] (+ 10 (* 5 ~dout[1])))
35  (def-delay /adrs[0] /dout[2] (+ 10 (* 5 ~dout[2])))
36  (def-delay /adrs[0] /dout[3] (+ 10 (* 5 ~dout[3])))
37  (def-delay /adrs[0] /dout[4] (+ 10 (* 5 ~dout[4])))
38  (def-delay /adrs[0] /dout[5] (+ 10 (* 5 ~dout[5])))
39  (def-delay /adrs[0] /dout[6] (+ 10 (* 5 ~dout[6])))
40  (def-delay /adrs[0] /dout[7] (+ 10 (* 5 ~dout[7])))
41  (def-delay /adrs[1] /dout[0] (+ 10 (* 5 ~dout[0])))
42  (def-delay /adrs[1] /dout[1] (+ 10 (* 5 ~dout[1])))
43  (def-delay /adrs[1] /dout[2] (+ 10 (* 5 ~dout[2])))
44  (def-delay /adrs[1] /dout[3] (+ 10 (* 5 ~dout[3])))
45  (def-delay /adrs[1] /dout[4] (+ 10 (* 5 ~dout[4])))
46  (def-delay /adrs[1] /dout[5] (+ 10 (* 5 ~dout[5])))
47  (def-delay /adrs[1] /dout[6] (+ 10 (* 5 ~dout[6])))
48  (def-delay /adrs[1] /dout[7] (+ 10 (* 5 ~dout[7])))
49  (def-delay /adrs[2] /dout[0] (+ 10 (* 5 ~dout[0])))
50  (def-delay /adrs[2] /dout[1] (+ 10 (* 5 ~dout[1])))
51  (def-delay /adrs[2] /dout[2] (+ 10 (* 5 ~dout[2])))
52  (def-delay /adrs[2] /dout[3] (+ 10 (* 5 ~dout[3])))
53  (def-delay /adrs[2] /dout[4] (+ 10 (* 5 ~dout[4])))
54  (def-delay /adrs[2] /dout[5] (+ 10 (* 5 ~dout[5])))
55  (def-delay /adrs[2] /dout[6] (+ 10 (* 5 ~dout[6])))
56  (def-delay /adrs[2] /dout[7] (+ 10 (* 5 ~dout[7])))
57  (def-delay /adrs[3] /dout[0] (+ 10 (* 5 ~dout[0])))
58  (def-delay /adrs[3] /dout[1] (+ 10 (* 5 ~dout[1])))
59  (def-delay /adrs[3] /dout[2] (+ 10 (* 5 ~dout[2])))
60  (def-delay /adrs[3] /dout[3] (+ 10 (* 5 ~dout[3])))
61  (def-delay /adrs[3] /dout[4] (+ 10 (* 5 ~dout[4])))
62  (def-delay /adrs[3] /dout[5] (+ 10 (* 5 ~dout[5])))
63  (def-delay /adrs[3] /dout[6] (+ 10 (* 5 ~dout[6])))
64  (def-delay /adrs[3] /dout[7] (+ 10 (* 5 ~dout[7])))
65  (def-delay /adrs[4] /dout[0] (+ 10 (* 5 ~dout[0])))
```

```
 66  (def-delay /adrs[4] /dout[1] (+ 10 (* 5 ~dout[1])))
 67  (def-delay /adrs[4] /dout[2] (+ 10 (* 5 ~dout[2])))
 68  (def-delay /adrs[4] /dout[3] (+ 10 (* 5 ~dout[3])))
 69  (def-delay /adrs[4] /dout[4] (+ 10 (* 5 ~dout[4])))
 70  (def-delay /adrs[4] /dout[5] (+ 10 (* 5 ~dout[5])))
 71  (def-delay /adrs[4] /dout[6] (+ 10 (* 5 ~dout[6])))
 72  (def-delay /adrs[4] /dout[7] (+ 10 (* 5 ~dout[7])))
 73  (def-delay /adrs[5] /dout[0] (+ 10 (* 5 ~dout[0])))
 74  (def-delay /adrs[5] /dout[1] (+ 10 (* 5 ~dout[1])))
 75  (def-delay /adrs[5] /dout[2] (+ 10 (* 5 ~dout[2])))
 76  (def-delay /adrs[5] /dout[3] (+ 10 (* 5 ~dout[3])))
 77  (def-delay /adrs[5] /dout[4] (+ 10 (* 5 ~dout[4])))
 78  (def-delay /adrs[5] /dout[5] (+ 10 (* 5 ~dout[5])))
 79  (def-delay /adrs[5] /dout[6] (+ 10 (* 5 ~dout[6])))
 80  (def-delay /adrs[5] /dout[7] (+ 10 (* 5 ~dout[7])))
 81  (def-delay /adrs[6] /dout[0] (+ 10 (* 5 ~dout[0])))
 82  (def-delay /adrs[6] /dout[1] (+ 10 (* 5 ~dout[1])))
 83  (def-delay /adrs[6] /dout[2] (+ 10 (* 5 ~dout[2])))
 84  (def-delay /adrs[6] /dout[3] (+ 10 (* 5 ~dout[3])))
 85  (def-delay /adrs[6] /dout[4] (+ 10 (* 5 ~dout[4])))
 86  (def-delay /adrs[6] /dout[5] (+ 10 (* 5 ~dout[5])))
 87  (def-delay /adrs[6] /dout[6] (+ 10 (* 5 ~dout[6])))
 88  (def-delay /adrs[6] /dout[7] (+ 10 (* 5 ~dout[7])))
 89  (def-delay /adrs[7] /dout[0] (+ 10 (* 5 ~dout[0])))
 90  (def-delay /adrs[7] /dout[1] (+ 10 (* 5 ~dout[1])))
 91  (def-delay /adrs[7] /dout[2] (+ 10 (* 5 ~dout[2])))
 92  (def-delay /adrs[7] /dout[3] (+ 10 (* 5 ~dout[3])))
 93  (def-delay /adrs[7] /dout[4] (+ 10 (* 5 ~dout[4])))
 94  (def-delay /adrs[7] /dout[5] (+ 10 (* 5 ~dout[5])))
 95  (def-delay /adrs[7] /dout[6] (+ 10 (* 5 ~dout[6])))
 96  (def-delay /adrs[7] /dout[7] (+ 10 (* 5 ~dout[7])))
 97  (def-delay /adrs[8] /dout[0] (+ 10 (* 5 ~dout[0])))
 98  (def-delay /adrs[8] /dout[1] (+ 10 (* 5 ~dout[1])))
 99  (def-delay /adrs[8] /dout[2] (+ 10 (* 5 ~dout[2])))
100  (def-delay /adrs[8] /dout[3] (+ 10 (* 5 ~dout[3])))
101  (def-delay /adrs[8] /dout[4] (+ 10 (* 5 ~dout[4])))
102  (def-delay /adrs[8] /dout[5] (+ 10 (* 5 ~dout[5])))
103  (def-delay /adrs[8] /dout[6] (+ 10 (* 5 ~dout[6])))
104  (def-delay /adrs[8] /dout[7] (+ 10 (* 5 ~dout[7])))
105  (def-delay /we /dout[0] (+ 10 (* 5 ~dout[0])))
106  (def-delay /we /dout[1] (+ 10 (* 5 ~dout[1])))
107  (def-delay /we /dout[2] (+ 10 (* 5 ~dout[2])))
108  (def-delay /we /dout[3] (+ 10 (* 5 ~dout[3])))
109  (def-delay /we /dout[4] (+ 10 (* 5 ~dout[4])))
110  (def-delay /we /dout[5] (+ 10 (* 5 ~dout[5])))
111  (def-delay /we /dout[6] (+ 10 (* 5 ~dout[6])))
112  (def-delay /we /dout[7] (+ 10 (* 5 ~dout[7])))
113  )
```

この PCD を用いて, KUE-CHIP2 で用いられるメモリをネットリスト (NLD) で, 実現します (リスト 7).

リスト 7: KUE-CHIP2 で用いられるメモリの実現 (r512_8.nld)

```
 1  (def-module r512_8
 2  (def-pin adrs[0] type input)
 3  (def-pin adrs[1] type input)
 4  (def-pin adrs[2] type input)
 5  (def-pin adrs[3] type input)
 6  (def-pin adrs[4] type input)
 7  (def-pin adrs[5] type input)
 8  (def-pin adrs[6] type input)
 9  (def-pin adrs[7] type input)
10  (def-pin adrs[8] type input)
11  (def-pin din[0] type input)
12  (def-pin din[1] type input)
13  (def-pin din[2] type input)
14  (def-pin din[3] type input)
15  (def-pin din[4] type input)
16  (def-pin din[5] type input)
17  (def-pin din[6] type input)
18  (def-pin din[7] type input)
19  (def-pin dout[0] type output)
20  (def-pin dout[1] type output)
21  (def-pin dout[2] type output)
22  (def-pin dout[3] type output)
23  (def-pin dout[4] type output)
```

```
24  (def-pin dout[5] type output)
25  (def-pin dout[6] type output)
26  (def-pin dout[7] type output)
27  (def-pin read type input)
28  (def-pin write type input)
29  (def-pin m_clock type input)
30  (def-pin s_clock type input)
31  (def-pin b_clock type input)
32  (def-pin p_reset type input)
33
34  (def-comp lpm_ram ram)
35  (def-comp and--2 and)
36  (def-comp inv- inv)
37
38  (def-net write and.in1)
39  (def-net read inv.in)
40  (def-net inv.nout and.in2)
41  (def-net and.out ram.we)
42  (def-net m_clock ram.m_clock)
43  (def-net s_clock ram.s_clock)
44  (def-net b_clock ram.in_clock ram.out_clock)
45  (def-net p_reset ram.p_reset)
46  (def-net adrs[0] ram.adrs[0])
47  (def-net adrs[1] ram.adrs[1])
48  (def-net adrs[2] ram.adrs[2])
49  (def-net adrs[3] ram.adrs[3])
50  (def-net adrs[4] ram.adrs[4])
51  (def-net adrs[5] ram.adrs[5])
52  (def-net adrs[6] ram.adrs[6])
53  (def-net adrs[7] ram.adrs[7])
54  (def-net adrs[8] ram.adrs[8])
55  (def-net din[0] ram.din[0])
56  (def-net din[1] ram.din[1])
57  (def-net din[2] ram.din[2])
58  (def-net din[3] ram.din[3])
59  (def-net din[4] ram.din[4])
60  (def-net din[5] ram.din[5])
61  (def-net din[6] ram.din[6])
62  (def-net din[7] ram.din[7])
63  (def-net dout[0] ram.dout[0])
64  (def-net dout[1] ram.dout[1])
65  (def-net dout[2] ram.dout[2])
66  (def-net dout[3] ram.dout[3])
67  (def-net dout[4] ram.dout[4])
68  (def-net dout[5] ram.dout[5])
69  (def-net dout[6] ram.dout[6])
70  (def-net dout[7] ram.dout[7])
71  )
```

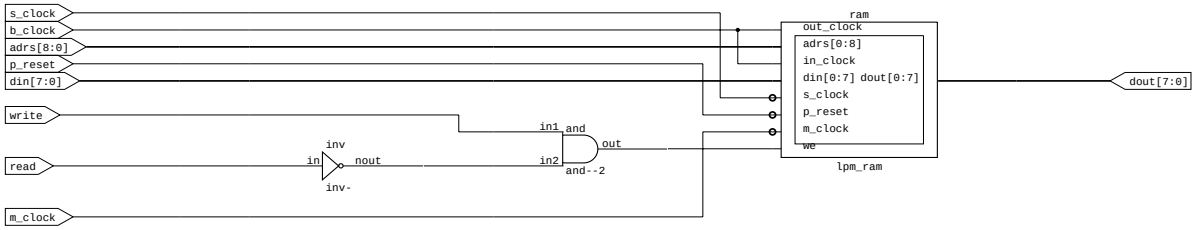

図 3: r512_8.nld

図 3はリスト 7 を図にしたものです. データ, アドレスの入出力ポートはそのまま直結しています. 読み書きの制御は KUE-CHIP2 の記述では write と read の 2 本の信号で制御しているのに対し, ALTERA では, we のみで制御を行っています. そのため INVERTER と 2 入力 AND を用いて, write が 1 かつ read が 0 の時にのみ we が 1 となるようにしています. また, クロックは ALTERA では, inclock, outclock の 2 本を用いますが, これは共通のクロックを入力することで十分ですので, 同一のクロックラインを接続します.

メモリの入出力のタイミング

ALTERA のデバイスの埋め込みメモリは前述のようにクロックラインを使用することにより同期動作を行います. この同期動作は基本的には非同期動作をするメモリの前後にレジスタを自動的に挿入することにより実現しています. そのため読み出し要求を行ってもそれに対応する値は同一クロック内には出力されません. 一方, KUE-CHIP2 は同一マシンサイクル (クロック) 内で値が出力されることと前提として記述されています. その違いを回避するためには, SFL 記述をメモリのタイミングに合わせて記述を変更するという方法が正当なやり方ですが, ここでは KUE-CHIP2 に投入するクロックとメモリに投入するクロックを分けるという方法を用いてこの違いを回避します.

PARTHENON のクロックラインについて

PARTHENON は基本的に単相同期のシステムですが, クロックのラインは m_clock, b_clock, s_clock, scan_clock の 4 本が存在します. PARTHENON では論理合成時に m_clock, s_clock, b_clock の 3 本のクロックラインを自動的に回路に付け加えられます. scan_clock はテスト付きの合成を行った時に付け加えられます. その後, 不要なクロックラインは削除されます. その判断はセルライブラリの中でそのクロックラインが使用されているかで行われています. エッジトリガ型のレジスタの場合, クロックラインは 1 本ですので, 通常は m_clock のみが使用されます. マスタスレーブ型の場合は 2 本使用しますので, m_clock, s_clock の 2 本が使用されます. b_clock は通常, バス用のクロックラインとして用いられます.

今回は KUE-CHIP2 のロジックに m_clock を, メモリに b_clock を用いて, b_clock に m_clock より高速なクロックを投入することにより, メモリからのデータの遅れを回避することにします. b_clock はバス用のクロックラインですが, どのラインがどのように接続されるかは上述のようにライブラリの記述に依存しますので, ライブラリの記述によってはこの様な用途にも使用することができます[6].

まずリスト 6 の 28 行目から 32 行目のように, メモリのクロックラインを設定します. lpm_ram_dq として, inclock, outclock を設定し, それ以外に s_clock, m_clock, p_reset を unused のピンとして設定します. 後者は前述のように合成の途中で削除されます. そして, 図 3の様に r512_8.nld で b_clock が inclock, outclock に接続されるように記述します (リスト 7, 44 行目). その他の入力も同様に接続します. この様にセルライブラリを記述することにより, メモリのクロックは b_clock により制御されるよう回路が合成されます.

3.3.2 MAX+plus Ⅱ でのマッピング

メモリの場合も, ALU と同様にライブラリマッピングファイル (リスト 8) を記述する必要があります.

リスト 8: par3.lmf

```
 1  LIBRARY par3
 2
 3  % lpm_ram_dq 512 8 %
 4  BEGIN
 5  FUNCTION lpm_ram_dq ( DATA0, DATA1, DATA2, DATA3, DATA4, DATA5, DATA6, DATA7,
 6                        ADDRESS0, ADDRESS1, ADDRESS2, ADDRESS3, ADDRESS4,
 7                        ADDRESS5, ADDRESS6, ADDRESS7, ADDRESS8,
 8                        WE, INCLOCK, OUTCLOCK)
 9  RETURNS  ( Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7)
10  FUNCTION "lpm_ram" ("din_0_", "din_1_", "din_2_", "din_3_",
11                      "din_4_", "din_5_", "din_6_", "din_7_",
12                      "adrs_0_", "adrs_1_", "adrs_2_", "adrs_3_", "adrs_4_",
13                      "adrs_5_", "adrs_6_", "adrs_7_", "adrs_8_",
14                      "we", "in_clock", "out_clock")
15  RETURNS  ("dout_0_", "dout_1_", "dout_2_", "dout_3_", "dout_4_", "dout_5_", "dout_6_", "dout_7_")
16  END
```

メモリの場合はライブラリのマッピングに加えて, LPM のパラメータを設定する必要があります. ここでは表 4に示されるパラメータを設定します. 設定は以下の手順で行います.

1. MAX+plus Ⅱ を起動し, File→Project→Name... を設定します.
2. Complier を起動します.

[6] もちろん, ライブラリの記述に誤りがあれば, 合成される回路も誤りが混入しますので, 注意が必要です.

3. Assign→Global Project Parameters... を開きます (図 4).

4. それぞれの値を設定します.

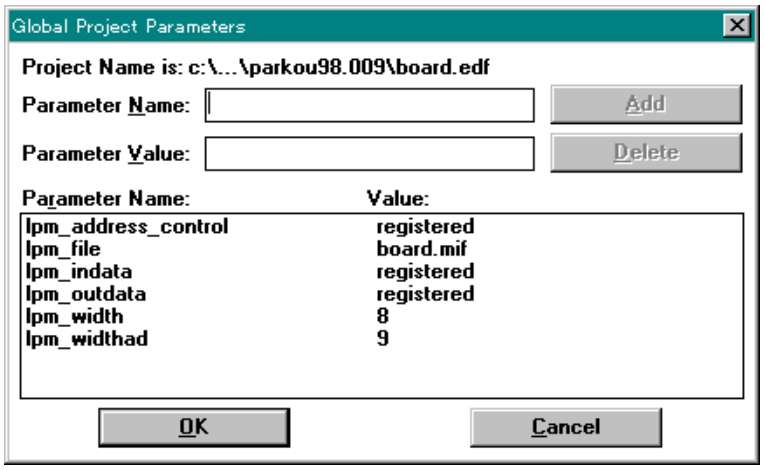

図 4: LPM のパラメータの設定

3.3.3 MAX+plus Ⅱ でのシミュレーション

MAX+plus Ⅱ はシミュレーションの機能も持っていますので, マッピングされたデータを元にタイミングシミュレーションが可能です. ここではリスト 9 のプログラムを実行して動作を確認します. このプログラムは 80 番地の値を N とし, 1 から N までの総和を計算し, 結果を 81 番地に格納するプログラムです.

リスト 9: 1 から N までの和を計算する KUE-CHIP2 のプログラム

```
 1  *** KUE-CHIP2 Assembler ver.1.0   by H.Ochi ***
 2
 3  * Sum from 1 to N
 4  * Programmed by Akira Uejima, May. 5, 1992
 5  * e-mail: uejima@mics.cs.ritsumei.ac.jp
 6
 7  * N
 8   80 :           N:      EQU             80H
 9  * Sum(result)
10   81 :           SUM:    EQU             81H
11
12   00 :   6C 80           LD      IX,     [N]
13   02 :   62 00           LD      ACC,    0
14   04 :   B1      LOOP:   ADD     ACC,    IX
15   05 :   AA 01           SUB     IX,     1
16   07 :   33 04           BP              LOOP
17   09 :   74 81           ST      ACC,    [SUM]
18   0B :   0F              HLT
19
20                          END
```

MAX+plus Ⅱ ではメモリの内容をあらかじめ設定しておくことが可能です. ここではプログラムはあらかじめ設定しておき, 80 番地の値はシミュレーション中に設定します. メモリの内容の設定は以下の手順で行います.

1. MAX+plus Ⅱ を起動し, File→Project→Name... を設定します.

2. Simulator を起動します.

3. Initialize→Initialize Memory... 開きます (図 5).

4. 値を設定します.

5. `Export File...` を押し, LPM のパラメータ, `lpm_file` で設定したファイル名で保存します. そのように設定するとシミュレーション時に自動的にメモリがそのファイルの内容で初期化されます.

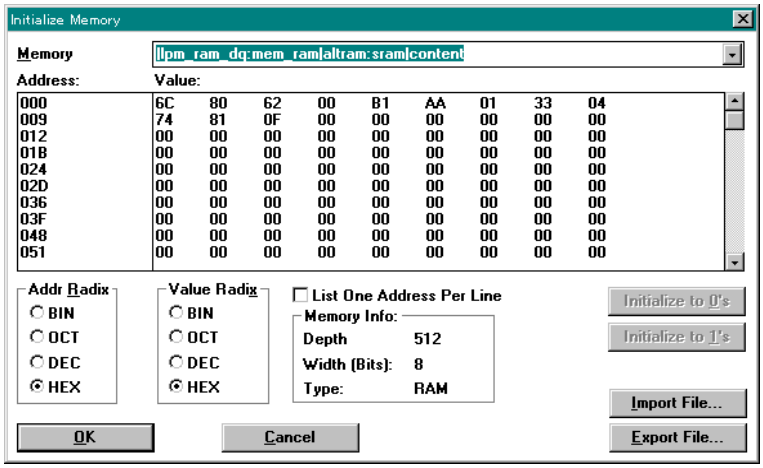

図 5: メモリの内容の設定

シミュレーションはまず `Graphic Editor` を用いて, 入力ピンへの値を設定し, 次に `Simulator` を動作させ, 再び `Graphic Editor` で, 出力ピンの様子を観察します. `Graphic Editor` の使用方法などは MAX+plus II のマニュアルなどを参照してください.

図 6はシミュレーション開始直後の様子, 図 7はシミュレーション終了時に様子を表わしています.

図 6では `m_clock`(KUE-CHIP2 へのクロック) と `b_clock` (メモリへのクロック) がそれぞれ別の速度で与えられている様子が分かります. シミュレーションの動作の内容は以下のようになります.

1. `p_reset` が 0 に設定し, 回路をリセットします. [7].
2. `mem_w_data_[7..0]_`に 0A, `mem_addr_[8..0]_`に 080 をそれぞれ設定し, `mem_w` をアクティブにすることにより, メモリの 80 番地に 0A を書き込みます.
3. `cpu_start` をアクティブにし, CPU を起動します.
4. プログラムが動作し, `HLT` 命令により停止します.

図 7では CPU が停止するところです. 1 から 0A(10 進数で 10) までの総和 37(10 進数で 55) が出力されている様子を見ることができます. また, 先ほどメモリ上にプログラムを設定したウインドウでもメモリに値が書き込まれた結果を見ることができます. 初期状態ではアドレスが 80, 81 ともに 00 ですが (図 8), シミュレーション後はそれぞれ 0A, 37 が書き込まれています (図 9).

4 終りに

今まで見てきたように, PARTHENON から FPGA 側に用意されているマクロやメモリなどを使用することできます. それらを使用する際にはそれらの特性に注意して SFL 記述を行うことや, ライブラリを作成することが大切です. 本稿が少しでも設計者の助けとなることを望みます.

参考文献

[1] *Data Book*, ALTERA, 1998.

[2] *ALTERA Home Page*, `http://www.altera.com/`.

[7] `p_reset` は low active です.

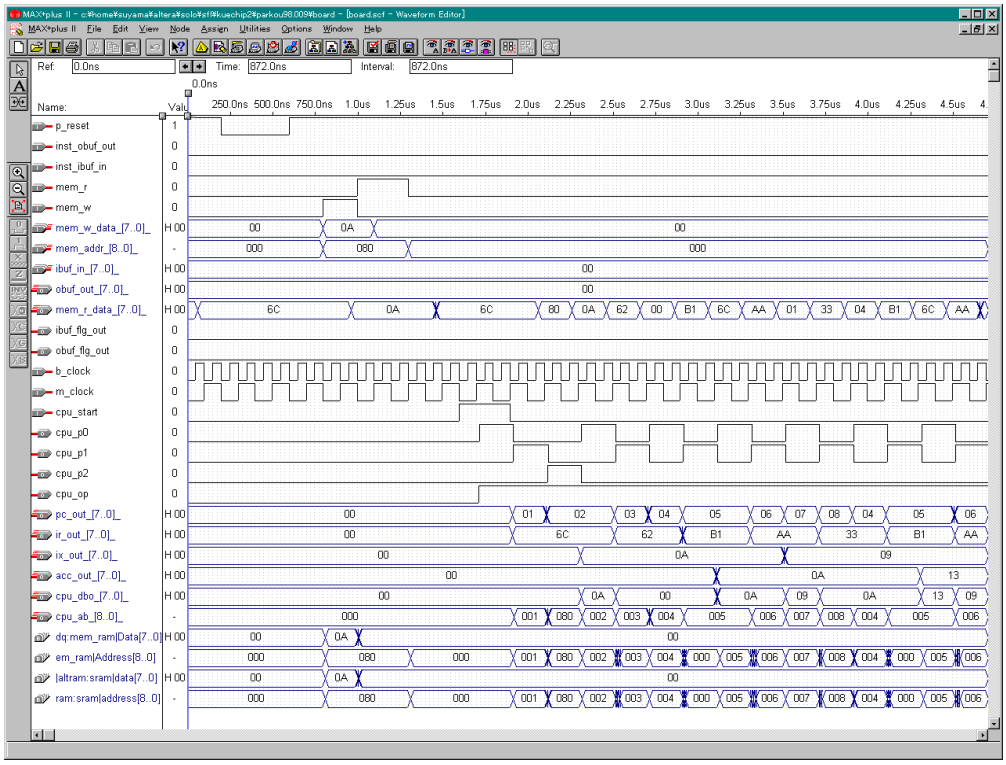

図 6: MAX+plus Ⅱによるシミュレーションの様子 (開始時)

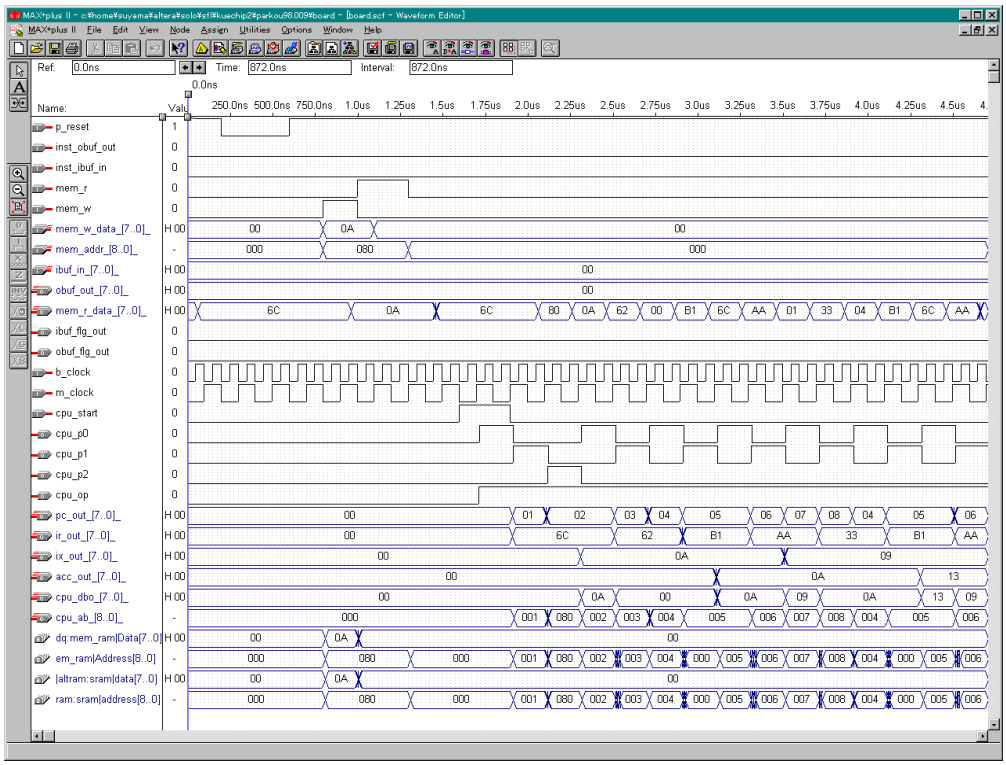

図 7: MAX+plus Ⅱによるシミュレーションの様子 (終了時)

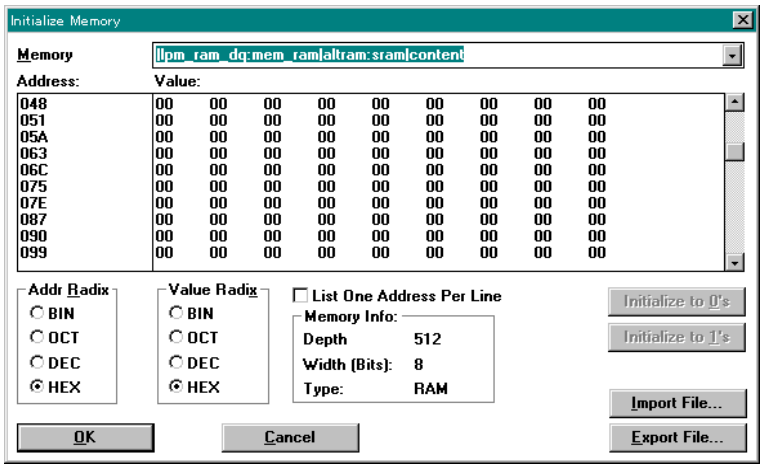

図 8: メモリの内容の確認 (シミュレーション前)

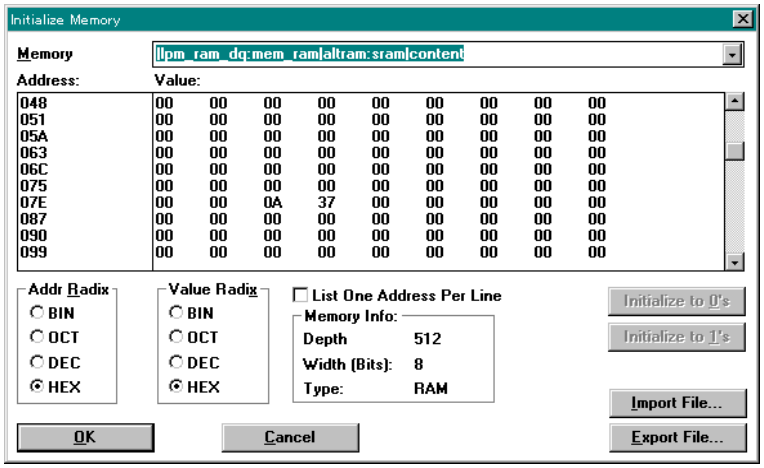

図 9: メモリの内容の確認 (シミュレーション後)

[3] *PARTHENON Home Page*, `http://www.kecl.ntt.co.jp/car/parthe/`.

[4] 澤田 宏: 8 ビット CPU (KUE-CHIP2) の設計, 第 5 回 PARTHENON 講習会テキスト, pp. 25–47, 1997.

[5] 須山 敬之: ALTERA FPGA への接続, 第 6 回 PARTHENON 講習会テキスト, pp. 145–152, 1998.

[6] *MAX+plusII Help File (for Windows95)*, ALTERA.