Preferred Networks インターン選考2019 コーディング課題 Chip分野

変更履歴

- 2019年4月19日 : 初版
- 2019年4月22日 : 課題2のスケジュール後アセンブリを訂正
- 2019年4月23日 :
  - 課題2に自明な繰り返しは省略して構わないことをを明記し、題意を明確化
  - 課題2の説明の文体を統一
  - 課題2の例を2次元配列に変更
  - **Performance Optimization分野の課題1をChip分野の課題2の代わりに解いていただいても結構です。**
    - したがって以下のいずれかの問題の組み合わせを選択してください。
      - Chip課題1, Chip課題2, Chip課題3
      - Chip課題1, Performance Optimization 課題1, Chip課題3

回答にあたっての注意

- 課題1ではソースコードには以下のいずれかの言語を利用してください。
  - SystemVerilog, Verilog HDL, VHDL
- 課題には自分だけで取り組んでください。この課題を他の応募者を含めた他人と共有・相談することを禁止します。課題期間中に GitHub の公開リポジトリ等にソースコードや問題をアップロードする行為も禁止します。漏洩の証拠が見つかった場合、その応募者は失格となります。ある応募者が別の応募者に回答をコピーさせた場合、双方の応募者が失格となります。
- 想定所要時間は最大2日です。全課題が解けていなくても提出は可能ですので、学業に支障の無い範囲で取り組んで下さい。

提出物

ソースコード(課題1)およびレポート(課題2-3)

評価基準

提出物は以下を満たしていることが望ましいです(必須ではありません)。

- 書かれた条件を満たしていること(課題1)
- ソースコードが読みやすいこと(変数名やインデント等)
- レポートが要点についてわかりやすくまとまっていること

一次選考後の面接で提出されたコードについて質問する可能性があります。

提出方法

上記の提出物を単一のパスワード無しzipファイルにまとめ、こちらの専用フォームより応募してください。締切は日本時間2019年5月7日（火）12:00です。

問い合わせ

問題文に関して質問がある場合はintern2019-admin@preferred.jpまでご連絡ください。問題文に訂正が行われた場合は応募者全員にアナウンスいたします。なお、アプローチや解法に関する問い合わせにはお答えできません。

問題文

課題1

Fig. 1の波形のように、アップカウント(信号inc_count)とダウンカウント(信号dec_count)を交互に行うようなテストベンチとRTL (Register Transfer Level) コードを記述して下さい。

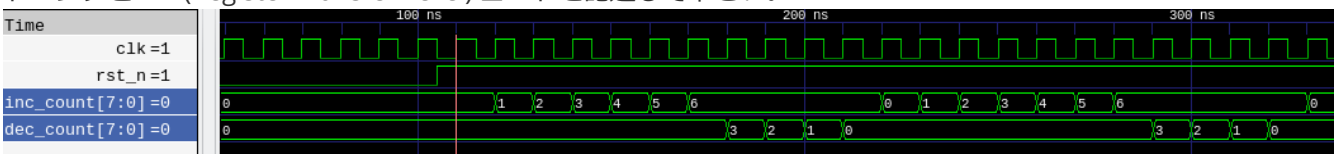

Fig. 1:期待する動作波形

使用言語とファイル名

SystemVerilog, Verilog HDL, VHDLのどの言語を使用しても構いません。言語のバージョンも任意です。 すべてのモジュールを単一のファイルにまとめて以下の通りのファイル名で保存して下さい。

HDL	ファイル名
SystemVerilog	q1.sv
Verilog HDL	q1.v
VHDL	q1.vhd

モジュール階層

Fig. 2のような階層構成にして下さい。
例)counter_topがモジュール名、カッコ内のi_counter_topがインスタンス名です。

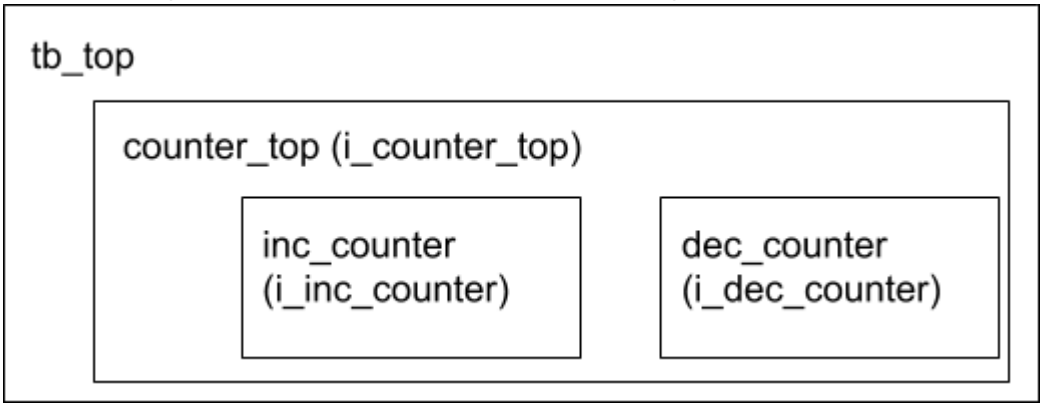

Fig. 2:モジュール階層

各モジュールの役割

各モジュールはそれぞれ以下のような役割をもたせて下さい。

モジュール名	説明
tb_top	クロック信号とリセット信号を生成する。このモジュールはテストベンチなので論理合成できる必要はありません。
counter_top	inc_counterモジュールとdec_counterモジュールをインスタンス・接続する。このモジュール以下は論理合成可能なRTL記述をして下さい。
inc_counter	リセット解除後動作を開始する。dec_counterの出力が0になると0から6までカウントする。
dec_counter	リセット解除後動作を開始する。inc_counterの出力が6になると3から0までカウントする。

内部信号

モジュールtb_top内には少なくとも以下の4つの信号を定義してください。それ以外の内部信号名は自由に名付けて下さい。

信号名	説明
clk	クロック信号
rst_n	リセット信号 Low Activeの非同期リセット
inc_count	inc_counterの出力 8bit信号でcounter_topから出力する
dec_count	dec_counterの出力 8bit信号でcounter_topから出力する

実行環境

もし手元にRTLシミュレーション環境がない場合はIcarus VerilogとGTK Waveのようなオープンソースソフトウェアをインストールする、もしくはEDA playground のようなWebサイトを利用してください。なお、これらのソフトウェアやWebサイトはPFNとは関係が無いため質問にお答えすることはできません。

課題2 (Performance Optimization分野の課題1との選択)

Fig. 3に示すような8レーンのSIMDアクセラレータ上で、行列aと行列bの乗算を行うためのプログラムを以下に示します。アクセラレータの命令セットを `命令セット` に示します。 Fig. 3(a)に示すように、アクセラレータは8個のローカルメモリとFunctional Unit(FU)のペアを持ちます。各ローカルメモリは64bit幅の共有データバスに接続されています。このデータバスは32bitのデータ(=1ワード)を同時に2つ転送できます。 Fig. 3(b)のようにローカルメモリは32bit幅なので、データバスの64bitのうちどちらの32bitを使うかマルチプレクサで選択できます。

1. `命令セット` に示す命令を使って、 `プログラム` に示す行列積の計算時間が短くなるようにしたアセンブリを記述してください。 `スケジュール後アセンブリ例` にもあるように、自明な繰り返しは省略して構いません。

2. アクセラレータの構成を1点だけ変更できるとき、どのような変更をすることで行列積の計算時間を更に短縮できるか検討し、例を挙げて理由を説明して下さい。

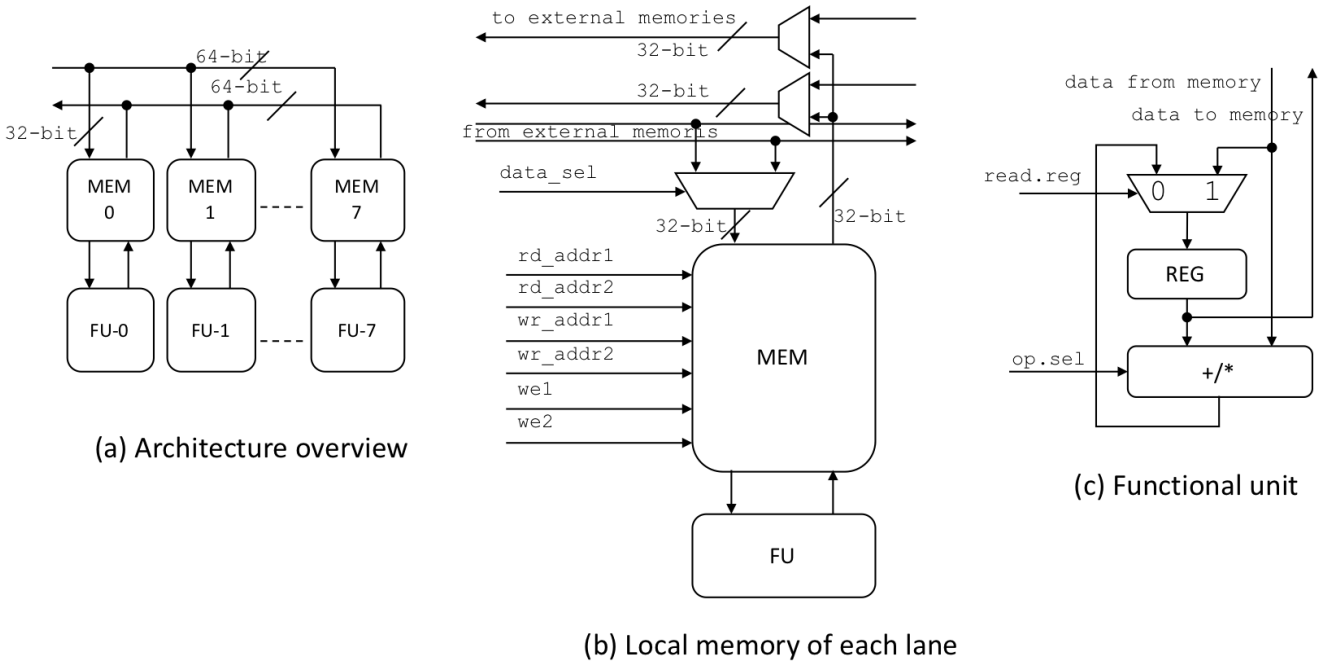

Fig. 3:アーキテクチャ

プログラム

```
#define N 32
int a[N][N], b[N][N], c[N][N];

void matrix_mult (int a[N][N], int b[N][N], int c[N][N]) {
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
      c[i][j] = 0;
      for (int k = 0; k < N; k++) {
        c[i][j] += a[i][k] * b[k][j];
      }
    }
  }
}
```

命令セット

外部メモリからローカルメモリへのデータ転送

`mov.a {a[row_idx0][col_idx],a[row_idx1][col_idx]}->{mem[lane0][addr],mem[lane1][addr]};`

外部メモリ上の行列a[N][N]の要素であるa[row_idx0][col_idx]とa[row_idx1][col_idx]の2ワードをローカルメモリにコピーします。ローカルメモリmemの最初のインデックスlane0、lane1はレーン番号を、2番目のインデックスaddrはローカルメモリ内のアドレスを意味します。

`mov.a {a[row_idx0][start_idx,len],a[row_idx1][start_idx,len]}->{mem[lane0][addr0,len],mem[lane1][addr1,len]};`

外部メモリのa[row_idx0][start_idx]～a[row_idx0][start_idx+len-1]とa[row_idx1][start_idx]～a[row_idx1][start_idx+len-1]をローカルメモリmemにコピーします。ローカルメモリmemの最初のインデックスlane0、lane1はレーン番号を、2番目のインデックスaddr0、addr1はローカルメモリ内のアドレスを意味します。

```
mov.b a[row_idx][col_idx]->mem[][addr];
```

外部メモリのa[row_idx][col_idx]をすべてのローカルメモリにブロードキャストします。コピー先が全レーンのローカルメモリなので、memの最初のインデックスは指定しません。memの2番目のインデックスaddrはローカルメモリ内コピー先アドレスです。

```
mov.b a[row_idx][col_idx,len]->mem[][addr,len];
```

外部メモリのa[row_idx][col_idx]～a[row_idx][col_idx+len-1]の内容を全レーンのローカルメモリにブロードキャストします。コピー先の開始アドレスはaddr、コピー長はlenです。

ローカルメモリから外部メモリへのデータ転送

```
mov.h {mem[lane0][addr],mem[lane1][addr]}->{a[row_idx0][col_idx],a[row_idx1][col_idx]};
```

ローカルメモリから外部メモリにデータをコピーします。memの最初のインデックスlane0とlane1はレーン番号です。 2番目のインデックスaddrはコピー元アドレスです。row_idx0とcol_idxおよびrow_idx1とcol_idxは外部メモリ上のコピー先アドレスです。

FUでの演算

以下の命令はすべてのFUで同時に実行されます。

```
read.reg mem[addr]; //reg = mem[addr]
```

ローカルメモリのアドレスaddrの内容をFunctional Unitのレジスタ REGにコピーします(Fig. 3 (c)参照)。

```
read.add mem[addr]; //reg = reg + mem[addr]
```

FUのレジスタREGとローカルメモリmem[addr]の内容の加算をします。結果はレジスタREGに書き込まれます。

```
read.mult mem[addr]; //reg = reg * mem[addr]
```

FUのレジスタREGとローカルメモリmem[addr]の内容の乗算をします。結果はレジスタREGに書き込まれます。

```
write mem[addr];
```

レジスタREGの内容をローカルメモリのアドレスaddrに書き込みます。

アクセラレータの制約

- FUとローカルメモリは8ペアあります。
- 各ローカルメモリの大きさは32bit × 128ワードで、リードとライトのポートがそれぞれ2つあります。
- Fig. 3(b)のようにライトポートのうち1つは共有データバスに接続されており、外部メモリ(External Memory)から受け取ったデータを書き込むために使われます。もう一方のライトポートはFUに接続されています。リードポートも同様にデータバスとFUに1ポートずつ接続されています。
- データ転送命令は1サイクルに1つしか発行できません。たとえば mov.aやmov.bとmov.hを同時に発行することはできません。
- mov.[a, b, h]とread/write命令は同時に発行できます。

- `mov.b a[row_idx][col_idx,len]->mem[][addr,len];` のように複数ワードの転送を行う命令はlenサイクルだけかかります。この期間は他のmov.[a, b, h]命令を発行できません。
- 各FUのレジスタ(Fig. 3(c)のREG)のサイズは1ワードです。
- このアクセラレータはSIMDアーキテクチャなので、moveとread/write命令は全レーンで同時に実行されます。(SIMD=Single Instruction Multiple Data)

例

2次元配列同士の加算

```
#define H    32
#define W    16
int v1[H][W], v2[H][W], r[H][W];
void array2d_add (int v1[H][W], int v2[H][W], int r[H][W]) {
  for (unsigned y = 0; y < H; ++y) {
    for (unsigned x = 0; x < W; ++x) {
      r[y][x] = v1[y][x] + v2[y][x];
    }
  }
}
```

データの分割

アクセラレータは8レーンあり、それぞれ128ワードのローカルメモリがあり入出力データを保存できます。 このベクトル加算の例では3 x 32 x 16 = 1536ワード必要になりますが、ローカルメモリは8 * 128 = 1024ワードしかありません。 したがって、まず配列を2ブロックに分割して、ローカルメモリあたりの各ブロックのサイズを3 * 32 x 16 / 2 / 8 = 96ワードにします。 分割されたデータは8レーンのローカルメモリに分配されます。 したがって以下のようにコードを変更できます。

```
#define H    32
#define W    16
int v1[H][W], v2[H][W], r[H][W];
void array2d_add (int v1[H][W], int v2[H][W], int r[H][W]) {
  for (unsigned i = 0; i < 2; ++i) {
    for (unsigned y = 0; y < H/2; ++y) {
      for (unsigned x = 0; x < W; ++x) {
        r[(i * H/2) + y][x] = v1[(i * H/2) + y][x] + v2[(i * H/2) + y][x];
      }
    }
  }
}
```

スケジュール後アセンブリ例

上記のような2次元配列の和を計算するアセンブリ例を以下に示します。

```
//256-word data transfer time from v1[0-15][0-15] is 128 cycles
mov.a  {v1[0][0, 16], v1[2][0, 16]} -> {mem[0][0, 16],mem[1][0, 16]};
mov.a  {v1[1][0, 16], v1[3][0, 16]} -> {mem[0][16, 16],mem[1][16, 16]};
mov.a  {v1[4][0, 16], v1[6][0, 16]} -> {mem[2][0, 16],mem[3][0, 16]};
mov.a  {v1[5][0, 16], v1[7][0, 16]} -> {mem[2][16, 16],mem[3][16, 16]};
mov.a  {v1[8][0, 16], v1[10][0, 16]} -> {mem[4][0, 16],mem[5][0, 16]};
```

```
mov.a  {v1[9][0, 16], v1[11][0, 16]} -> {mem[4][16, 16],mem[5][16, 16]};
mov.a  {v1[12][0, 16], v1[14][0, 16]} -> {mem[6][0, 16],mem[7][0, 16]};
mov.a  {v1[13][0, 16], v1[15][0, 16]} -> {mem[6][16, 16],mem[7][16, 16]};


//256-word data transfer time from v2[0-15][0-15] is 128 cycles
mov.a  {v2[0][0, 16], v2[2][0, 16]} -> {mem[0][32, 16],mem[1][32, 16]};
mov.a  {v2[1][0, 16], v2[3][0, 16]} -> {mem[0][48, 16],mem[1][48, 16]};
----
mov.a  {v2[12][0, 16], v2[14][0, 16]} -> {mem[6][32, 16],mem[7][32, 16]};
mov.a  {v2[13][0, 16], v2[15][0, 16]} -> {mem[6][48, 16],mem[7][48, 16]};


//first 32 words computation time 96 cycles
//iteration 0    (3 cycles)
read.reg    mem[0];
read.add    mem[32];
write        mem[64];

//iteration 1 (3 cycles)
read.reg    mem[1];
read.add    mem[33];
write        mem[65];
----
//iteration    31
read.reg    mem[31];
read.add    mem[63];
write        mem[95];

//32 results are transferred from local memories to external memory
mov.h  {mem[0][64], mem[1][64]}->{r[0][0], r[2][0]};
----
mov.h  {mem[0][79], mem[1][79]}->{r[0][15], r[2][15]};
mov.h  {mem[0][80], mem[1][80]}->{r[1][0], r[3][0]};
----
mov.h  {mem[0][95], mem[1][95]}->{r[1][15], r[3][15]};

----

mov.h  {mem[6][64], mem[7][64]}->{r[12][0], r[14][0]};
----
mov.h  {mem[6][79], mem[7][79]}->{r[12][15], r[14][15]};
mov.h  {mem[6][80], mem[7][80]}->{r[13][0], r[15][0]};
----
mov.h  {mem[6][95], mem[7][95]}->{r[13][15], r[15][15]};



//similarly the remaining computation are performed as follows:
mov.a  {v1[16][0, 16], v1[18][0, 16]} -> {mem[0][0, 16],mem[1][0, 16]};
mov.a  {v1[17][0, 16], v1[19][0, 16]} -> {mem[0][16, 16],mem[1][16, 16]};
----
mov.a  {v1[28][0, 16], v1[30][0, 16]} -> {mem[6][0, 16],mem[7][0, 16]};
```

```
mov.a  {v1[29][0, 16], v1[31][0, 16]} -> {mem[6][16, 16],mem[7][16, 16]};


//256-word data transfer time from v2[0-255] is 128 cycles
mov.a  {v2[16][0, 16], v2[18][0, 16]} -> {mem[0][32, 16],mem[1][32, 16]};
mov.a  {v2[17][0, 16], v2[19][0, 16]} -> {mem[0][48, 16],mem[1][48, 16]};
----
mov.a  {v2[28][0, 16], v2[30][0, 16]} -> {mem[6][32, 16],mem[7][32, 16]};
mov.a  {v2[29][0, 16], v2[31][0, 16]} -> {mem[6][48, 16],mem[7][48, 16]};


//first 32 words computation time 96 cycles
//iteration 0    (3 cycles)
read.reg    mem[0];
read.add    mem[32];
write        mem[64];

----
//iteration    31
read.reg    mem[31];
read.add    mem[63];
write        mem[95];


//32 results are transferred from local memories to external memory
mov.h  {mem[0][64], mem[1][64]}->{r[16][0], r[18][0]};
----
mov.h  {mem[0][79], mem[1][79]}->{r[16][15], r[18][15]};
mov.h  {mem[0][80], mem[1][80]}->{r[17][0], r[19][0]};
----
mov.h  {mem[0][95], mem[1][95]}->{r[17][15], r[19][15]};

----

mov.h  {mem[6][64], mem[7][64]}->{r[28][0], r[30][0]};
----
mov.h  {mem[6][79], mem[7][79]}->{r[28][15], r[30][15]};
mov.h  {mem[6][80], mem[7][80]}->{r[29][0], r[31][0]};
----
mov.h  {mem[6][95], mem[7][95]}->{r[29][15], r[31][15]};
```

課題3

1. あなたの好きなコンピューターアーキテクチャを1つ選んで、好きな理由を説明して下さい。 (別にないときは任意のアーキテクチャを選んで、なぜ選んだかを説明して下さい。 (最大250字)
2. 紹介したアーキテクチャの長所と短所を説明して下さい。 (最大250字)
3. コンピューターアーキテクチャと消費電力の関係を説明して下さい。 (最大250字)