吉本研究室

computational science and computer science

第6回課題

課題評価方法

- 6回の課題は、個人ごとに行い、また6回と7回の2回にわたって実施する。第7回の 17:00以前に、シミュレーションによる検証に合格することおよびVivadoで正常に合成できたこととその遅延時間測定の結果を、評価者に点検してもらう。そして第7回の週の金曜日 17:00 以前に、 課題ごとに別に作成したVerilogのmoduleをすべて収納した単一のファイル、遅延時間の測定に使ったVivadoのプロジェクト一式（プロジェクトのディレクトリの外にあるファイルも忘れないこと）、およびオプション課題のレポート(提出するなら)をITC-LMSに課題ファイルとして提出する。ITC-LMSにアップロードするファイルは、姓と学生証番号を使ってkadai6_yosimoto_05201099.zipのように名前をつけた物とし、提出ファイルをzipないしtgz形式のアーカイブにまとめた物とすること。課題ごとに別に提出するVerilogのファイルは、その課題の検証コードに改変なしに接続できなければならない。
  成果物の点検は、原則シミュレーションによる検証によって機械的に行う。シミュレーションを通過しないものはできていないものと解釈するので注意すること。
- 第7回は、今回に引き続いてこの課題を指導者の支援を受けながら行い、完成したら評価を受ける回になる。必須課題の評価は班員全員が提出で合意しslackに書き込んだ班から順に班員全員分をまとめて行う。オプション課題はこれとは別に個人ごとに評価を行うが、評価の順は提出順ではなく、巡回しやすい順とする。

Vivadoのシミュレーション機能を用いて、以下の課題を実行せよ。

1. 符号なし32bit数dと、符号なし64bit数であって上位32bitがd未満の数xについて、除算x = q*d + rを組み合わせ論理で計算し、32 bit数qとrを出力するmoduleを作成しシミュレーションで検証せよ。検証にはtest_divider.svを改変せずに利用せよ。なお、作成に当たっては”/”演算子と”%”演算子を利用しないこと。また除算のIPコアも利用しないこと。test_divider.svは作成したモジュールの名前が「div32」であることを期待しているので注意せよ。test_divider.svは、除算結果をチェックし、問題があればその箇所を指摘する。従って指摘がなければ合格である。指摘のフォーマットは以下の様になっている。

```
q "被除数" "除数" "作成したモジュールによる商" "正解の商"
r "被除数" "除数" "作成したモジュールによる余り "正解の余り"
```

ゼロによる除算の定義：test_divider.svは0による除算の結果、商には’1(＝2進数としてbit数個ならんだ1)が、余りには被除数そのものが出力されることを期待しているので注意すること。

2. 1の回路のクリティカルパスを2分割することで、レイテンシ2で1クロック1演算のスループットを持つ順序回路として構成し、シミュレーションで検証せよ。検証にはtest_divider_p2.svを改変せずに利用せよ。test_divider_p2.svは作成したモジュールの名前が「div32p2」であることを期待しているので注意せよ。さらに実際に合成、配置配線して動作可能周波数が約２倍になることを確認せよ。この確認は、クリティカルパスの遅延時間をそれぞれns単位で報告し、この2分割によって遅延時間が約半分になることを確かめることで行うこと。確認にはこのアーカイブに含まれるVivadoのプロジェクトを使うこと。
3. オプション課題：dがメルセンヌ数8191である場合に、符号なし32bit数xの除算x = q*d + rをできるだけ小規模な回路でかつ短い遅延時間で計算して、qとrを出力するmoduleを作成し、シミュレーションで動作を検証せよ。検証にはtest_div_8191.svを改変せずに利用せよ。test_div_8191.svは作成したモジュールの名前が「divider_8191」であることを期待しているので注意せよ。test_div_8191.svは、除算結果をチェックし、問題があればその箇所を指摘する。従って指摘がなければ合格である。指摘のフォーマットは以下の様になっている。

```
y "被除数" "正解の余り(2進)" "計算した余り(2進)"
z "被除数" "正解の商(2進)" "計算した商(2進)"
```

また提出するレポートではどのように設計したか説明すること。
なおレイテンシを問題にしているので組み合わせ論理で作ること。課題1と同様に”/”演算子と”%”演算子は利用しないこと。また除算のIPコアも利用しないこと。合成した回路の規模は全LUT通算(LUT1からLUT6)で評価せよ。遅延時間も測定して報告すること。測定は課題2のアーカイブに入っているプロジェクトで使っている方法と同様の方法で行うこと。

課題1,2のための検証ファイルはここからダウンロードせよ。除算のアルゴリズムはこちらを参照せよ。オプション課題3のための検証ファイルはここからダウンロードせよ。

また、以下は課題のための参考である。

コマンドラインでのシミュレーションの実行方法

1. WindowsメニューのXilinxのフォルダから、Vivado用のコマンドプロンプトを起動する。これで必要なパスの設定などがなされたコマンドプロンプトが起動する。
2. 前提：シミュレーションしたいデザインファイル(Verilogなどによるファイル)一式だけが入ったディレクトリを用意し、このディレクトリをカレントディレクトリとして作業を行う。D:ドライブを作業データ用とし、C:ドライブはシステムソフトやアプリケーションソフト用と切り分けるのが良い。パス名に決して日本語などASCII文字でないものを使わないこと。途中もダメ。
3. デザインファイルを解析し、解析済みデータをカレントディレクトリ以下のxsim.dirディレクトリ以下に保存する。構文エラーはこの段階で分かる。テストベンチを担当するtop moduleがあり、timescaleを指定しているファイルtopfile.v (topfile.sv)を先頭に指定する。

```
$ xvlog topfile.v file1.v file2.v
$ xvlog --sv topfile.sv file1.sv file2.sv # SystemVerilog
```

4. デザインスナップショットのエラボレートおよび生成。
   スナップショットとはシミュレータ本体が実行できる形式でモデルを表記したもの。
   最上位ユニットtop_of_testのスナップショットをtop_of_test.simとして生成する例。

```
$ xelab -debug typical top_of_test -s top_of_test.sim
```

5. シミュレーションを実行

```
$ xsim -runall top_of_test.sim
$ xsim top_of_test.sim -gui # exec with GUI
```

6. GUIを使っている場合には、シミュレーションのステップ実行と波形の確認ができる。
   1. 波形を観察したい信号をObjectsウィンドウのリストで選択して右クリックし、Add to Wave Windowする。複数bit幅の信号の場合、その中の一部の信号を選択することも出来る。
   2. Run メニューの中のRun All、Run for、Restartや、メニュー直下にある対応するボタンを使って、シミュレーションを動かすと、波形が出てくる。波形は1であらかじめ指定した信号のみ記録されるので、追加した場合には最初からシミュレーションを行わなければならない。
   3. 複数bitの波形にはその値が表示されているが、この表示方法は、波形のウィンドウにあるNameに表示されている信号の名前の上で右クリックすると出てくるRadixを使うことで、変更できる。実数表示させることもRadix->Real Settingsから指定できる。
7. GUIを使わずにシステムタスク$dumpと$dumpfileを使ってシミュレーションを行い、波形の記録を一度にとって、それを後でubuntuにインストールしたgtkwaveなどの波形表示ソフトで観察することも出来る。

Vivadoの使い方

- 作成したblock diagramのmoduleとして、作成したHDL designを引用する方法
  1. Block Designを開いておく
  2. DesignのSourcesパネルから、Design Sourcesを右クリックして出てくるAdd Sourcesでファイルを取り込む
  3. Diagramの上で右クリックして出てくる、Add Moduleでblock diagramに追加する
- SourcesのDesign Sourcesのtopが意図している物になっているか注意すること
  - projectを作る際にHDLのfileを取り込むとそれがtopになることがある。しかし取り込んだファイルがBlock Diagramで使う予定のmoduleの場合は意図どおりではない。
  - Block Diagramに対応するwrapperを作らせて、それをtopに指定しないとおかしい事になる。
- block diagram上のmoduleとして、直接引用できるHDL designはVerilogによるものかVHDLによるものに限られており、SystemVerilogは直接引用できない。そのため、一旦Verilogによるwrapperを書いて、そこからSystemVerilogのモジュールを引用する必要がある。source fileとしてはwrapperおよびSystemVerilogで書いた本体の両方をDesign Sourcesに追加する。
- implementationを実行する際にImplementation Settingsでopt_designが有効になっていると、外部に信号を取り出さないdesignは最適化によって全体が除去される事がある。したがって、あらかじめダミーで良い

のでLEDなどに出力信号をつなぐこと。Utility IPのreductionで一旦信号を集合させると良い。この際、一部の信号が固定的にHまたはLになることで他の信号の意味が失われることを避けるためxorを使うこと。

- 設計した回路のslackを手に入れるためには、前後にレジスタをつないでやる必要がある。RAM based shift registerなどでこれを作る。レジスタの値を保持したければ、自分自身に出力を戻せば良い。初期値の設定はpower on resetの設定でできた。
- slackの確認はOpen Implemented DesignやProject Summaryからできる。どこからどこまでのslackが最悪になっているか確認すること。最悪遅延時間の逆数が動作可能周波数を決める。（clock skewの問題が多少ある）
    - Worst Negative Slackが最悪のslackを表している。これが正であれば、タイミング解析に合格する回路の合成に成功したことになる。Open Implemented Designすると出てくるDesign Timing Summaryに表示されているWorst Negative Slackをクリックすると対応するcritical pathの情報が得られる。
- Block Design上の各モジュールが使ったFPGAの資源量(LUTの数など)は、SynthesisしたときのReports, Synthesis, Out-of-Context module Runsの、自分が作ったmoduleのUtilization Reportの中にある。

Vivadoのバグ(2018.3まで存在していた。2019.2ではなくなっている)

x[32:0], d[31:0]としたとき、x>=dが正しくシミュレータに評価されない。代わりにx[31:0]>=dが評価されてしまう模様。x[33:0],d[32:0]とx[31:0],d[30:0]では問題ない。

Verilogによるデザインのデバッグのヒント

Verilogは言語仕様が「緩い」ために、通常は書かない（書くことが推奨されない）記述をしても、処理系がエラーやワーニングを出さないことがある。実際Vivadoのシミュレーターはかなり寡黙である上、合成したらエラーになるような記述をしても、エラーにしないことがある。そのため、記述のチェックに「verilator」を活用することを推奨する。verilatorの本来の使い方はシミュレーターとしてでであるが、誤りと思われる記述を見つける機能(lint)が充実しており、この機能だけに限定して使うことが出来る。ただし、verilatorは合成可能なVerilogの記述のみを扱うことが出来るものなので、テストベンチを含めて点検することは出来ず、作成したモジュールだけ点検することになる。例えば「myexample.v」に記述したVerilogのモジュールを点検するには、

```
verilator --lint-only -Wall myexample.v
```

とコマンドを使えば良い。System Verilogを使っている場合には、

```
verilator --lint-only +1800-2012ext+sv -Wall myexample.sv
```

とコマンドを使う。なお、verilatorのインストールは各種パッケージから行うことが出来るが、かなり頻繁にバージョンアップされているので、自分でコンパイルしたものを使うとより良いだろう。コンパイルとインストールは簡単である。