～ハードウェア設計論シケプリ～

文責：大和田

一応大事そうなところをまとめましたが、これが授業の内容にどの程度沿ってるかはよく分かりません。何人かに聞いてみても、何やってたかよく分からないような感じなので…。まああまり勉強する気のない人は目を通しておいて損はないと思います。きっと何とか単位くらいは。

レイアウトは見ての通りなので、読みにくいのはご勘弁。内容が薄いのは、うｐが遅くなったためみんな今からそんなに読む気がしないだろうという配慮で、決して何書けばいいか分からなかったからではありませんよ、ハイ。では前置きはこのくらいで。

～藤田先生の内容～

１．　ハードウェア・ソフトウェア協調設計について

組込みシステムでは、ハードウェア・ソフトウェアを平行して設計・開発して最適な機器を実現する必要があります。かつてはこの二つの設計はバラバラで、分割も HW 技術者の勘と経験で行われていましたが、システム規模の増大に伴ってこの設計法は限界を迎えました。そこで重視されるようになってきたのが**コデザイン(CO-Design)**の考え方です。

コデザインに必要な環境は二つあって、一つは HW/SW の**最適な分割の探索**をする環境、もう一つは HW/SW の**協調設計**環境です。

最適な分割の探索とは、いくつかの分割パターンにおけるコスト（面積）、性能、消費電力の**見積り**を行い、どの分割がいいかを検討することです。これには、見積りの精度が高くなければなりません。見積りの精度が低く、下流工程まで行ったときにやっぱりこの分割じゃダメだー、というのは最悪です。よい分割を見つけたら、CPU（ソフト）とハードを結ぶ**インターフェース（I/F）**を作成し、協調設計環境を実現します。

・I/F 合成

モジュール（HWorSW）間の接続部分を自動生成する処理です。HW-SW の場合についてみると、

①メモリ以外の HW にもアクセスできるよう、アドレスデコーダを合成して HW に接続
②割り込み信号を自動接続
③HW にアクセスするためのドライバを自動生成

の三段階からなります。

これにより HW/SW 協調設計・検証が可能になります。結局のところ協調設計とは HW・SW それぞれの機能のシミュレーションを協調させて実行させることで、同時に設計・検証していくことです。協調設計は様々な抽象レベルで行われますが、抽象レベルによって異なる言語を使う必要があるとそこでミスが生じやすくなり、そもそも HW と SW で異なる言語を使っていたら書き換えの際のミスも生じやすくなります。そのため、HW と SW の両方を記述可能で、様々な抽象レベルのモデルを記述可能な言語が必要になります。これを満たす言語として、SystemC,C/C++, SpecC などがあります。C/C++は SW 用の言語ですが、ライブラリ等を用いることで抽象度の高い記述が可能になります。

ちなみに、協調設計と言っても HW/SW はあくまでもそれぞれの基準で評価する必要があります。HW は精度、SW は速度です。この結果が悪ければ分割からやり直します。

また、協調設計のメリットとして、HW を待たずとも SW の設計・開発が出来るため、全体の工程が短くなるという点が挙げられます。

２．システムレベル設計について

これについては過去問で触れたのでここでは一言だけ触れます。

システムレベル設計は、上流工程を定量的に、正確な言語で記述していくのが特徴です。その後の HW/SW 分割・協調設計がうまくいくかどうかを決める大事なポイントです。

３．ハードウェアとソフトウェアの特徴

これについても過去問で触れたので簡単に。とにかく文字通り HW はハードで SW はソフトです。ハードウェアは一度設計したら変更が出来ませんが、その分特定の仕様に特化すれば SW よりもはるかに高速な処理が可能になります。SW はその柔軟性がウリで、最後の最後でも仕様変更が出来ますが、処理は低速です。いかにシステムの一部をうまくハードウェア化するかが HW/SW 協調設計における分割のカギです（多分）。

４．BDD と SAT 手法

BDD については、VLSI 設計工学 2002-2 に、SAT については SAT の資料にそれぞれ載っているので、それを眺めるといいかもしれません。というかここには図を描かないので、そちらを参照することをお薦めします。

BDD（Binary Decision Diagram：2 分決定グラフ）と SAT 手法（Satisfiability Check：充足可能性判定手法）は、**論理等価性の検証**の際に使用されます。設計の段階が進んだ時、前段階の回路なり記述なりと論理的に等価かな？という検証です。

まず以前使われていた BDD ですが、これは真理値表を 2 分決定木に表すというものです。0 なら左、1 なら右。同じ遷移はまとめます。Apply 演算という操作で、2 つのグラフの AND や OR も取れます。

何故 BDD が論理等価性の検証に使えるかというと、等価な論理で入力変数の順番が同じならば、必ず同じグラフ形式に変換できるためです。ただし、入力変数の順番によってサイズが大きく異なるため、最もコンパクトな形にするためには、入力変数を最適な順番で並べてやる必要があります。しかしこの最適な順番を見つけるのは実は NP 完全問題なので、いつもコンパクトな形にできるとは限りません。したがって、論理等価性の検証が終わらないケースもあります。

最近使われるようになったのが SAT 手法です。これは、和積形の論理式について、出力を 1 にするような入力の組が存在するかを解くことです。例えば、論理式として A xor B を考えると、これは A と B が異なる時に 1 を出力するので、もしこの SAT の解が見つかれば、A と B は等価ではないことになります。これにより論理等価性の検証を行うことができます。

ただし SAT 手法も NP 完全問題なので、最悪の場合は検証が終わらない…のかな？　明記されてませんが多分そうでしょう。

二つの手法を比較すると、BDD は全ての解を見つけるもので、SAT は解を一つだけ見つけるものです。得意不得意が異なるので、うまく組み合わせて使うと効果的らしいです。

ちなみに、これらの数学的な手法はいずれも**形式検証**といい、シミュレーションとは別のものです。

～池田先生の内容～

01．イントロ

重要な箇所については過去問やこのシケプリで大体触れていますが、全体の様々な概念がチラホラと載っているので、軽く目を通しておくといいかもしれません。

FPGA について一言補足。FPGA は何がすごいかというと、ハードウェアなのに利用者が仕様を書き換えられるということです。

03．Verilog

これは正直どうしたもんですかね。結局演習もなかったし、多分出ないんじゃないかと思いますが。VerilogHDL の特徴は、とにかく全部書かなければならないことです。線も記述しないとダメです。一応確認しておきたい人は、大津君作のコンピュータアーキテクチャのレポート対策プリントを見ておくといいと思います。

設計フロー

大切なのはデジタル LSI の設計フロー図です（そのまんま）。あとはどうでもいいと思います。この設計フロー図ですが、用語で覚えようとすると混乱する可能性が高い（過去問でもそうでしたが、フロー図の構造はそれぞれの図によって結構異なる）ので、それぞれの段階で LSI がどういう形で記述をされているかの遷移を把握しておくといいかもしれません。簡単に並べると、

C 言語とか→VerilogHDL とか→演算器とか→ゲートとか→実際の配線とか→製造！

という感じになります。

ハードウエアの故障とテスト

テストや検証に関する問題は何かしら必ず出題されると思います。テストにどんな種類があるかを眺めておくといいと思います。特性を調べるテストについては深いところまで突っ込まれることはないでしょう。過去問で触れましたが、故障診断は要注意。可能性の高い故障パターンを作ってその故障が起きているか調べる、ということをやっています。個々の故障の名前とかは知らなくていいと思います。ちなみにプリント 3 ページ目の例題 1 のところの故障表というのがよく分からないんですが、これ誰か分かりませんかね。まあディジタル回路的な問題が出たりするのかは不明ですが。

一通り眺めてみましたが、配点が高いと噂の池田先生のほうで果たしてどんな問題が出るのでしょうね？　どう考えても藤田先生の範囲のほうが話題満載で問題も作りやすい気がするんですけど。

だいぶやっつけっぽくなってしまいましたが、テスト頑張りましょう！