CS294-48 Scribe Notes

Greg Gibeling
UC Berkeley
gdgib@eecs.berkeley.edu

October 1, 2009

1 Introduction

These notes are roughly organized by the slide on which the discussion took place, and include what I could cull as the highlites. I attempted not to include any long discussions, but only the consensus understanding at the end.

2 Graphical Representation

2.1 Machine Vocabulary

Difference between networks and channels is that networks are specialized units. Channels are point-to-point, but they may be complex internally (Ethernet). All four entities can contain active logic and lots of it. The diagram is for HUMAN representations, though the actual circuits involved can vary greatly.

2.2 Hierarchy Within Unit

Probably need graphical syntax to represent ports (Greg suggests Click). A pattern includes a problem, and perhaps a graphical pattern as the output. For example when we have multiple independent inputs and want to stride them across processing units, the pattern on this slide generally applies.

2.3 Transaction Scheduler

The concepts Transaction Scheduler, Controller and Sequencer are related, and may be the same. How do we graphically represent a transaction scheduler? The problem is that a scheduler is a temporal entity, not a structural one, and this is a structural graphic, so how do we represent scheduling?

Below are different graphical representations:

Regions Dashed regions with a distinguished controller, a la Ptolmey. Short hand for one unit which broadcast to all units in the region

Explicit Dashed or gray lines which denote control message connections.

Units Make controllers just simple units all to themselves. Represent them with diamonds as in flow-chart choices.

Structural hierarchy implies control scope, i.e. each hierarchically defined unit has it's own controller (at least commonly in a well drawn representation)

2.4 Leaf-Level Hardware

What's a pattern vs what's vocabulary

3 Hardware Patterns

3.1 Decoupled Units

3.1.1 Slide1

The first discussion is about the pattern itself. A better definition of decoupling: when two units only interact via handshake, or when they have no assumed timing for their interactions. Pattern does not say whether buffering is bounded or not. Anti-pattern would be to build something which requires unbounded buffering. Could add to pattern with how to do decoupling for certain instances, but this isn't a requirement of the pattern itself (may be a subpattern). What happens if you need to design a unit which is externally decoupled and internally coupled?

The second discussion was about pattern hierarchies and relationships. There should be some structure to the pattern language, but no one can ever agree on what it is. As an example I remember an OPL/Arch meeting spend about an hour arguing terminology with this to little effect.

3.1.2 Slide2

Not all networks are decoupled, a network could be statically scheduled. If the units are decoupled the

network/memory must be decoupled as we don't know when units will be ready for the shared resources. Decoupling is transitive, anything connecting to a decoupled unit must be decoupled.

Can implement decoupling by:

Multiporting A port is statically scheduled to a consumer, but they may or may not use it.

Valid tagged data The unit provides a dummy word of data which is tagged as "not really data."

3.2 Pipelined Operator

Increases throughput and latency both. Costs resources. Controller requires the addition of a shift register to track when the operator is done. Figure is backwards.

3.3 Multicycle Operator

Decreases issue rate and throughput, increases/decreases latency (depending on operator), decreases cycle cost. There is a triangle of parameter/-tradeoff between latency, throughput and issue rate. self-timed/async logic is all "multicycle" to some extent, simply because it's universally decoupled with no integer-multiple-of-clock-cycles requirements. Helps with power, can help with latency by removing pipeline registers (when compared to pipeline register).

3.4 No Slide

Pipeline and multicycle patterns solve related patterns, we should start looking at how to group them perhaps. For each of the BHPL patterns Krste outlined, we can probably write a pattern very much like what's on these slides. Synchronous design vs Self-Timed are definitely patterns For each group of patterns (e.g. memory implementation or operator implementation) we can probably write a table or a decision tree which lists all the patterns and the associated benefits of each in a prescriptive way.

3.5 Control+Datapath

What the heck is data vs what is control? This turned into a large discussion and at least some of the salient points are below. To start with we do care about the distinction: errors in the control are hardware to catch, and have very high impact (*e.g.* error in packet payload is low impact vs header can cause dropped connection).

Control tends to be higher entropy, data is lower entropy. Perhaps the difference is that we care about the bits in the datapath, but we only care that the controller is somewhat sensible. Control is select inputs to all the muxes (and load enables)? There is a continuum of control vs datapath, e.g. in self-timed from more control to more data: handshake, control tokens, data values.

Control is contained within a unit. The control for one level is data to the lower level. Two ways to decompose a unit (each unit is an FSM): either hierarchically or control/datapath.

For any unit you should be able to describe it's functionality, and as a combination of datapath and control, though control may be degenerate (e.g. combinational adder). This forces the designer to at least deliberately decide to use the degenerate case, making them less likely to accidentally hide the control in with the datapath.

Pretty much all controllers are conditional on one input: Reset (though there are self-reseting). We'll probably end up with a taxonomy of controllers, which explains all the different styles of controllers (FSM, shift register, counter, etc).

Control and datapath are part of the vocabulary, but the transaction scheduler style of controllers is probably a pattern.