Automatic Pipeline Synthesis in Chisel

Huy Vo, Wenyu Tang
University of California, Berkeley
{huytbvo, wenyu}@eecs.berkeley.edu

1. Introduction

There have been great productivity gains in digital design from using hardware description languages such as Verilog and VHDL to specify digital circuits. Further productivity gains be achieved by increasing the abstraction level in which digital designers describe hardware. The approach we take in this paper is to separate the description of data flow within a datapath, which we will now refer to as datapath specifiction, from the the description of how to pipeline that datapath, which we will now refer to as pipelining specification, and then automatically synthesize the pipelined datapath. Manually pipelining a datapath is timing consuming and error prone because the designer has to insert non-trivial control logic that becomes increasingly complex with increased pipeline depth. By separating the specification of the data flow from the specification of the pipelining, we allow the designer to more easily explore the design space of pipeline depth and pipeline hazard resolution options.

In this paper we leverage Chisel's existing functionality as a wiring language do the datapath specification, which consists of specifying a set of architectural state elements and their next state logic, and extend Chisel to allow for a separate pipelining specification, which includes pipeline depth and how to resolve each pipeline harzard that is present in the design, and the automatic synthesis of the pipelined datapath. We support resolving each pipeline hazard with one of three options: interlocking, bypassing, and speculation.

[**?**]

2. Specification

2.1 Chisel

We utilize Chisel(Constructing Hardware In a Scala Embedded Language) as the basis of our datapath and pipelining specification. Chisel is an experimental hardware description language implemented through a set of class definitions within the high level programming language Scala. Chisel allows for the designer to fully leverage the high level object oriented programming features of the Scala language to succinctly describe hardware. A design described in Chisel can be mapped to a C++ simulator or Verilog emitted for either FPGA simulation or ASIC synthesis.

2.2 Datapath Specification

The datapath can be specified by a set of architectural state elements and their next state update logic. We also add a special Variable Latency Interface to allow users to integrate into the datapath functional units that do not have valid output available every cycle, such as caches and multi-cycle divders.In the context of our specification, Chisel's existing syntax is used to wire up combinational logic and modules implementing the Variable Latency Interface, which functions as next state logic, to Reg and Transaction Mem constructs, which function as a single architectural state element and an addressable array of architecural state respectively.

Regs are existing Chisel constructs that represent a single multi-bit wide flip-flop with synchronous reset and clock enable ports. They can be treated as single piece of architectural state that is conditionally updated by every transaction contingent on the clock enable input.

Transaction Mems are Chisel Components that wrap exisiting Chisel Mem constructs in an IO interface that helps the automatic synthesis tools detect bypassing opportunities. On instantiation the Transaction Mem allows the user to specify the number of lines in the memory, number of read ports on the IO, number of virtual write ports on the IO, and the number of physical write ports on the underlining Chisel Mem construct. Each read port consists of a read address input and a read data output. There is a one to one mapping between Transaction Mem read ports and underlining read ports in the Chisel Mem construct. Each virtual write port consists of a write address input, write enable input, and write data input. The Transaction Mem wrapper automatically maps the virtual write ports on the IO to the physical write ports of the underlining Chisel Mem construct. If the number of virtual write ports exceeds the number of physical write ports, the Transaction Mem module automatically coalesces multiple virtual write ports onto a single physical write port by muxing together the write datas and write addresses of the virtual write ports and ORing together the write enables of the virtual write ports. The user should wire each distinct source of write data along with the associated write enable and write address into separate virtual write ports rather than manually coalescing the write datas together with a mux and wiring the output of the mux into a single virtual write port in order to allow the automatic sythesis tools to generate fine grained bypass logic. This will be elaborated upon in the section discussing bypass logic generation. *Remember to insert picture later*

Variable Latency Interfaces are a Chisel Components that specify a set of user facing IO to allow users to integrate a existing variable/multi-cycle latency Chisel module into the datapath specification and a set of tool facing IO to allow the automatic synthesis tools to generate appropriate pipeline logic to deal with variable/multi-cycle latency modules. The Variable Latency Interface exposes on its user facing IO a request ready input, request data input, and response data output. The user connects the user facing IO to the rest of the datapath using standard Chisel syntax. The tool facing IO includes a request ready output and a response valid output. The tools use the request ready output to generate logic that puts back pressure on the pipeline stages before the variable latency module and use the response valid output to generate logic to insert bubbles into the pipeline stages after the variable latency module. Figure xx gives and example of how to integrate a Data Cache into the datapath specification through wrapping it in a Variable Latency Interface. *Remember to insert picture later*

Everything used in the datapath specification is either a built in Chisel construct(Regs and combinational logic) or a ordinary Chisel Component(TransactonMems and Variable Latency Module Interfaces). This allows the user to create a datapath specification in the exact same way they would write an ordinary Chisel module, with the exception that they have to use TransactionalMems in place of Mem constructs and have to wrap up funtional units that do not have valid output data every cycle in Variable Latency Interfaces. The datapath specification by itself, without any pipelining specification, functions as a single cycle implementation of the datapath.

2.3 Pipelining Specification

. How to describe pipelines.

3. Automatic Pipeline Synthesis

3.1 Inserting Pipelines

3.2 Stage Coloring

3.3 Hazard Discussion

Hazard Types.

Detection.

3.4 Hazard Resolution Options

Interlocks.

Speculation.

Bypassing.

4. Results

4.1 Sodor

4.2 Comparison

5. Conclusion

References