Facilitating the Specification and Implementation of Pipelined Designs with Skeletor

Ivan Rodriguez-Ferrandez^{1,2}, Javier Barrera^{1,2}, Jeremy Giesen^{1,2},
Alvaro Jover-Alvarez ^{1,2}, Leonidas Kosmidis^{2,1}

^{1}Universitat Politècnica de Catalunya (UPC) ^{2}Barcelona Supercomputing Center (BSC)

***Abstract*—Skeletor is an open source EDA tool which reduces the bootstrap effort of new hardware designs. Skeletor facilitates the design specification as well as its automatic translation to the hierarchy of Verilog files, testbenches, synthesis and simulation scripts. In its 1.5 version we added support for the pipelined designs. In this paper, we describe the newly introduced feature and show its benefits with two real-world examples.**

I. Introduction

New hardware design projects come with a high associated start up effort cost, which increases the entry bar of inexperienced hardware designers and decreases the productivity of more seasoned designers. In particular, a new hardware design project requires the creation of the source code files containing the hierarchy of the components which make-up the design, along with their defined interfaces, translated from the design specification. As a matter of fact, based on our previous experience reported in [1], the amount of code contained in such files can measure up to 50% of the overall code of the design, with only 50% of the code actually corresponding to the implementation of the hardware logic. Moreover, a new project needs the creation of testbenches for the test-driven verification of each individual component or group of components, as well as the top level design, and their integration with synthesis and simulation tools.

Skeletor [1] is an open source EDA tool [2] which facilitates the definition of the specification of a hardware design and more importantly, it automates the tedious and error prone translation of the design specification to the *skeleton* of Verilog source code and testbench files. In this way, it helps to create unambiguous, well-documented design specifications which do not rely on developer's interpretation of the specification and allows hardware designers to focus only on what matters, the implementation of the design. Skeletor is under active development and currently supports both open source and proprietary synthesis and simulation environments and includes several user-friendly features.

The design specification can be described either in textual form using the Skeletor specification language which is similar to Verilog with C++ extensions [1], or in a visual way using a powerful integration with the hierarchical schematics supported by the popular open source EDA tool KiCad [3]. In the former case, syntax highlighting for the Skeletor language has been added in popular editors and automatic KiCad schematic generation is supported, for the documentation of the design. In the latter case, that is if the user opts for the visual specification of the design using KiCad, the design specification in the Skeletor description language is automatically generated, without the need to learn its syntax at all.

In this paper we describe a new feature added in the latest 1.5 version of Skeletor, which facilitates the specification of pipelined designs and allows their cleaner implementation. In addition, it can be used to migrate non-pipelined designs to pipelined ones with minimal changes. We provide concrete examples showing the use and the advantages of this new feature, which are provided in our open source repository as artifacts for their tutorial value as well as for reproducibility purposes.

II. Pipelined Designs

Pipelining is a widely used technique in hardware design, which is mainly applied in order to reduce the critical path of a circuit, so that a higher clock frequency can be achieved. In its most basic form, the implementation of this technique consists of inserting a flip-flop along the critical path of the design, effectively splitting it in two paths of combinational logic with lower timing constraints than the original one.

Unfortunately, timing information is not available at the design specification time, therefore pipelining is frequently introduced in a design during the implementation as a part of an iterative process, which increases the complexity and impacts the readability of the code. Thankfully, according to a recent study [4], modern CAD tools are capable of automatically implementing this optimisation, without the need of an explicit description from the developer.

In addition to the improvement of the clock time, when circuit pipelining is combined with explicit partitioning of a design, it can allow additional architectural optimisations such as higher utilisation from overlapping of operations in time, reusing a functional unit across different operations, or even turning off parts of a design when they are not used. Unlike the previous example, in this case the implementation of such features i.e. the pipeline flip-flops have to be present in the design specification from the beginning, which again complicates the logic implementation and understanding.

Skeletor, in its latest 1.5 version release, supports the *flop* type specifier in the connections between modules, which allows to define pipeline flip-flops in the design specification description. This way, the pipeline flip-flops are explicitly

```
wire  U3.wreg -> U5.wreg;
wire  U4.m2reg -> U3.m2reg;
wire  U4.wreg -> U3.wreg;
wire  U3.m2reg -> U5.m2reg;
wire  U4.wmem -> U3.wmem;
wire [BITS_REGFILE:0] U4.destination -> U3.destination;
wire [AddrSize-1:0] U4.aluresult -> U3.aluresult;
wire [AddrSize-1:0] U4.op2 -> U3.op2;
wire [AddrSize-1:0] U5.datareg -> U2.datareg;
wire [AddrSize-1:0] U1.instruction -> U2.instruction;
wire [AddrSize-1:0] U2.op1 -> U4.op1;
wire [AddrSize-1:0] U2.op2 -> U4.op2;
wire [AddrSize-1:0] U2.extendedimm -> U4.extendedimm;
wire [AddrSize-1:0] U3.dmemout -> U5.dmemout;
wire [BITS_REGFILE:0] U3.destination -> U5.destination;
wire [AddrSize-1:0] U3.aluresult -> U5.aluresult;
wire  U5.wreg -> U2.wreg;
wire [BITS_REGFILE:0] U5.destination -> U2.destination;
wire  U1.nextpc -> U1.pc;
wire  U2.aluc -> U4.aluc;
wire  U2.aluimm -> U4.aluimm;
wire  U2.wreg -> U4.wreg;
wire  U2.wmem -> U4.wmem;
wire [BITS_REGFILE:0] U2.destination -> U4.destination;
wire  U2.m2reg -> U4.m2reg;
```

Figure 1: Excerpt of module connections in a Single Cycle Processor Datapath Specification written in the Skeletor specification language.

```
flop  U3.wreg -> U5.wreg;
flop  U4.m2reg -> U3.m2reg;
flop  U4.wreg -> U3.wreg;
flop  U3.m2reg -> U5.m2reg;
flop  U4.wmem -> U3.wmem;
flop [BITS_REGFILE:0] U4.destination -> U3.destination;
flop [AddrSize-1:0] U4.aluresult -> U3.aluresult;
flop [AddrSize-1:0] U4.op2 -> U3.op2;
flop [AddrSize-1:0] U5.datareg -> U2.datareg;
flop [AddrSize-1:0] U1.instruction -> U2.instruction;
flop [AddrSize-1:0] U2.op1 -> U4.op1;
flop [AddrSize-1:0] U2.op2 -> U4.op2;
flop [AddrSize-1:0] U2.extendedimm -> U4.extendedimm;
flop [AddrSize-1:0] U3.dmemout -> U5.dmemout;
flop [BITS_REGFILE:0] U3.destination -> U5.destination;
flop [AddrSize-1:0] U3.aluresult -> U5.aluresult;
flop  U5.wreg -> U2.wreg;
flop [BITS_REGFILE:0] U5.destination -> U2.destination;
flop  U1.nextpc -> U1.pc;
flop  U2.aluc -> U4.aluc;
flop  U2.aluimm -> U4.aluimm;
flop  U2.wreg -> U4.wreg;
flop  U2.wmem -> U4.wmem;
flop [BITS_REGFILE:0] U2.destination -> U4.destination;
flop  U2.m2reg -> U4.m2reg;
```

Figure 2: Excerpt of module connections in a Multi-Cycle, Pipelined Processor Datapath Specification written in the Skeletor specification language.

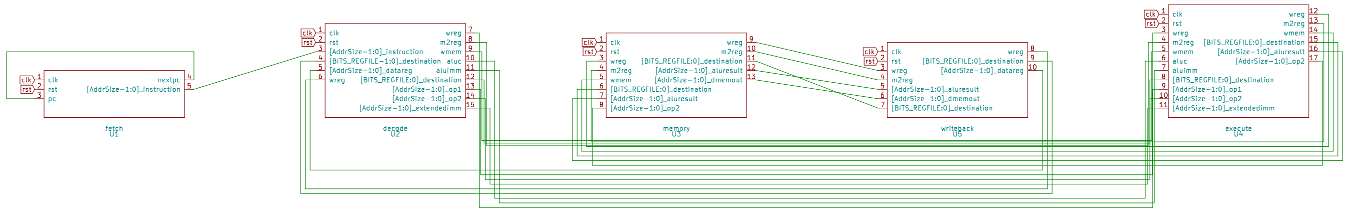


Figure 3: Automatically generated KiCad schematic of the Single Cycle datapath of the processor design of Figure 1.

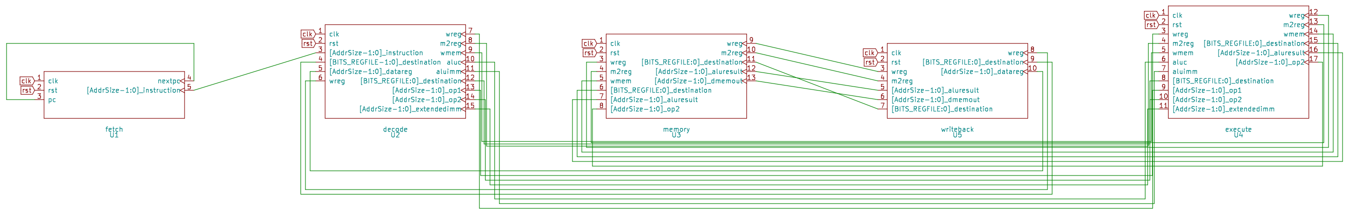


Figure 4: Automatically generated KiCad schematic of the Multi-Cycle, pipelined datapath of the processor design of Figure 2. Notice the positive edge-triggered flip-flop symbol ($\boxed{\wedge}$) in the connection between the signals in the modules.

documented in the design specification, and their skeleton code is automatically generated by Skeletor as positive edge-triggered D flip-flops, eliminating possible human errors such as wrongly specified sensitivity list, which is common in entry level developers. Moreover, this feature minimises the code to be written, since Skeletor takes care about the additional wire definitions (of the Verilog *reg* type) and their connections. Note that the generated Verilog code for the flip-flop skeleton does not include an enable signal, so if this is required by the implementation, it is up to the developer to introduce it.

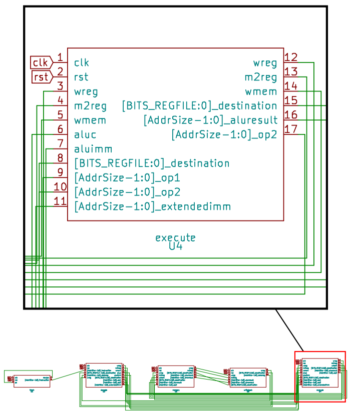
Although the primary focus of this feature is the explicit pipeline design at the design specification time, it can also be used for the iterative pipeline design which was mentioned at the beginning of the section. In that case, with appropriate decomposition of the design in the module hierarchy, a data path can be converted from a single cycle implementation to a pipelined one with minimal changes. Moreover, a fine-grained module decomposition in the specification, allows the designer to experiment with various placements of pipeline flip-flops, optimising the retiming of the design with minimal effort.

Next, we provide two examples which show how *flop* is used for both aforementioned cases.

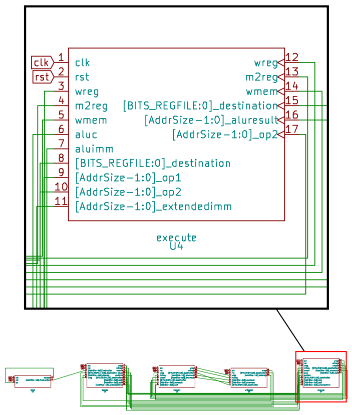
III. Pipelined Processor Datapath Example

In this example, we show how the data path design specification of a simple RISC-V processor can be converted from a single cycle design to a pipelined one.

Figure 1 shows an excerpt of the Skeletor specification of the simple processor design used in [1], which specifies the connections between the modules comprising the processor



(a) Close-up of Figure 3.



(b) Close-up of Figure 4.

Figure 5: Close-up of KiCad generated schematics.

```
module execute(AddrSize,BITS_REGFILE){
    in  clk;
    in  rst;
    in  wreg;
    in  m2reg;
    in  wmem;
    in  aluc;
    in  aluimm;
    in [BITS_REGFILE:0] destination;
    in [AddrSize-1:0] op1;
    in [AddrSize-1:0] op2;
    in [AddrSize-1:0] extendedimm;
    out  wreg;
    out  m2reg;
    out  wmem;
    out [BITS_REGFILE:0] destination;
    out [AddrSize-1:0] aluresult;
    out [AddrSize-1:0] op2;

    execute_1:U41(){
        in  rst = in rst,
        in  clk = in clk,
        in  wreg = in wreg,
        in  m2reg = in  m2reg,
        in  wmem = in  wmem,
        in  aluc = in aluc,
        in  aluimm = in aluimm
    };

    execute_2:U42(){
        in  rst = in rst,
        in  clk = in clk,
        out wreg = out wreg,
        out m2reg = out m2reg,
        out wmem = out wmem
    };

    wire  U41.wreg -> U42.wreg;
    wire  U41.m2reg -> U42.m2reg;
    wire  U41.wmem -> U42.wmem;
    wire  U41.aluc -> U42.aluc;
    wire  U41.aluimm -> U42.aluimm;
}
```

Figure 6: Single execution stage specification.

```
module execute(AddrSize,BITS_REGFILE){
    in  clk;
    in  rst;
    in  wreg;
    in  m2reg;
    in  wmem;
    in  aluc;
    in  aluimm;
    in [BITS_REGFILE:0] destination;
    in [AddrSize-1:0] op1;
    in [AddrSize-1:0] op2;
    in [AddrSize-1:0] extendedimm;
    out  wreg;
    out  m2reg;
    out  wmem;
    out [BITS_REGFILE:0] destination;
    out [AddrSize-1:0] aluresult;
    out [AddrSize-1:0] op2;

    execute_1:U41(){
        in  rst = in rst,
        in  clk = in clk,
        in  wreg = in wreg,
        in  m2reg = in  m2reg,
        in  wmem = in  wmem,
        in  aluc = in aluc,
        in  aluimm = in aluimm
    };

    execute_2:U42(){
        in  rst = in rst,
        in  clk = in clk,
        out wreg = out wreg,
        out m2reg = out m2reg,
        out wmem = out wmem
    };

    flop  U41.wreg -> U42.wreg;
    flop  U41.m2reg -> U42.m2reg;
    flop  U41.wmem -> U42.wmem;
    flop  U41.aluc -> U42.aluc;
    flop  U41.aluimm -> U42.aluimm;
}
```

Figure 7: Pipelined execution stage with two stages.

datapath. In this design, the connections between modules are performed using the *wire* specifier. Note that the color code of the listing matches the syntax highlighting template we include for the sublime Text3 Editor [5]. Figure 3 shows the automatically generated KiCad schematic of the datapath from the Skeletor specification.

Figure 2 shows the corresponding Skeletor specification excerpt for the same processor datapath, but with pipeline flip-flops between its different modules, which now comprise the datapath's pipeline stages. Notice that the only difference between the specifications is the replacement of the *wire* type specifier in the connections with the *flop* type specifier.

Figure 4 shows the automatically generated schematic for the pipelined version of the specification. Notice that again the only difference between the schematics is the presence of the positive edge-triggered flip flop ( ) between the module connections, as it can be seen better in the close-up of the two figures shown in Figure 5.

Obviously, the two designs have different control paths, which are part of their implementation and therefore they are not visible in the hierarchy specification descriptions, neither in their corresponding schematics. In fact, the specification shown in Figure 4 can be identical for both a multi-cycle or fully pipelined processor, with the only difference in the logic of their control path, assuming the absence of bypasses. This is the main advantage of explicit pipeline specification design; while as already mentioned modern CAD tools could automatically insert pipeline flip-flops in order to increase the maximum frequency of the processor, the resulting design will be a multi-cycle processor but not a fully pipelined design as considered by an architect, in which the execution of multiple instructions are overlapped in time. The reason is that this requires changes in the control path in order to stall the pipeline in case of hazards, something that current CAD tools lack the intelligence to do automatically.

IV. Processor Retiming Example

In this example, we consider the case of an existing design already specified in Skeletor, where the designer wants to increase its operational frequency. Let's assume that after the timing analysis of the processor datapath we examined in the previous Section, the critical path of the design has been identified in the processor's Execution stage, which contains only combinational logic.

Figure 6 shows the specification of this stage, which consists of two modules. By modifying the connection between these two modules using the *flop* type specifier instead of *wire* as shown in Figure 7, allows to seamlessly introduce a new pipeline stage, in order to reduce the critical path.

This operation can be repeated after a new timing analysis is performed, inserting new pipeline stages between the various modules and modifying if needed the control circuit of the datapath, until the target frequency is reached.

V. Related Work

Skeletor's support for the easy definition of pipelines is not a unique feature. To our knowledge, support for the seamless integration of pipelines in HDL has been added first within R&D activities at Intel Corporation. This work has been later open sourced and resulted in the TLV-Comp open source tool [6], maintained by TL-X.org, a special interest group for extensions of HDLs for higher levels of abstraction, including timing and transactions. The SandPiper tool [7] from Redwoord EDA is a commercial version of the TLV-Comp which supports the latest version of the TL-Verilog specification (Transaction-Level Verilog). All these tools, currently support only System Verilog similar to Skeletor.

Both TL-X tools and Skeletor support a similar rich set of functionalities, such as integration with editors, visualisation capabilities etc. However, their purpose is completely different, so an apples-to-apples comparison between them is not applicable. TL-Verilog is a full replacement of HDL languages with integrated timing abstraction targeting design implementation. On the other hand, Skeletor focuses on earlier stages of hardware design, mainly on the design specification and some automation regarding project bootstrapping.

In terms of pipelining support, the main difference between Skeletor and TL-Verilog is that TL-Verilog allows the definition of pipeline stages anywhere within the logic implementation of a module file. However, Skeletor is limited to the introduction of pipeline registers only between connections of modules, not their implementation. That is, the Skeletor assisted retiming possibilities for a design are limited to the finer grained modules existing in the design hierarchy.

On the other hand, Skeletor is a fully open source tool with active development. TLV-Comp has not received any updates since its first release in March 2017, while SandPiper is constantly maintained according to the latest advances in the TL-X specification definitions, but it is not open source and it is free only for open-source developments.

VI. Conclusion

We described the introduction of the *flop* type specifier in the version 1.5 of Skeletor, which facilitates the definition and implementation of pipelined designs. We have shown its benefits with 2 examples, one regarding the conversion of single cycle design to a pipelined one and one about logic retiming of a design for achieving higher frequency. Finally, we related this functionality with similar EDA tools.

Acknowledgments

This work is partially supported by the Spanish Ministry of Science and Innovation (MINECO) under grant PID2019-107255GB and the HiPEAC Network of Excellence. Leonidas Kosmidis is also funded by MINECO under a Juan de la Cierva Formación postdoctoral fellowship (FJCI-2017-34095). The authors would like to thank Guillem Cabo from BSC for his technical and theoretical support and prof. Roger Espasa from Universitat Politècnica de Catalunya, Esperanto Technologies and Semidynamics, for his useful feedback regarding Skeletor and his suggestions for the inclusion of the pipeline feature.

References

[1] I. Rodriguez, G. Cabo, J. Giesen, J. Barrera, A. Jover, and L. Kosmidis, "Skeletor Connector Language: Hierarchy Specification to HDL Development Made Easy," in *Workshop on Open-Source EDA Technology (WOSET)*, November 2019.

[2] ——, "Skeletor," Sep. 2019. [Online]. Available: https://github.com/jaquerinte/Skeletor

[3] Jean-Pierre Charras et al., *KiCad Complete Reference Manual*. 12th Media Services. [Online]. Available: www.kicad-pcb.org

[4] Steve Hoover, "Retiming Study with TL Verilog." [Online]. Available: https://github.com/stevehoover/warp-v/blob/master/doc/retiming.md

[5] Sublime HQ, "Sublime Text." [Online]. Available: https://www.sublimetext.com

[6] TL-X.org, "TLV-Comp." [Online]. Available: https://github.com/ypyatnychko/tlv-comp

[7] Redwood EDA, "SandPiper." [Online]. Available: https://www.redwoodeda.com/products