ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

**Mary Sheeran, Chalmers Technical University, Sweden, and Satnam Singh,
University of Glasgow, UK (Eds)**

Designing Correct Circuits

Proceedings of the 3rd Workshop on Designing Correct Circuits
(DCC96), Båstad, Sweden, 2-4 September 1996

ISBN: 3-540-76102-0

Paper:

Improving Hardware Designs Whilst Simplifying their Proof

Paul Curzon and Ian Leslie

Published in collaboration with the
British Computer Society

Improving Hardware Designs whilst Simplifying their Proof

Paul Curzon

Computer Laboratory, University of Cambridge,
Cambridge, UK

Ian Leslie

Computer Laboratory, University of Cambridge,
Cambridge, UK

Abstract

We describe a study to investigate the notion of “design for verifiability”. Our aim was to determine whether the cost of verification by interactive proof can be reduced by making appropriate design decisions. A major consideration was that such decisions should not compromise other design goals such as performance and functionality. We attempted to formally verify a real hardware design, noting factors which slowed the proof. On the basis of the identified bottlenecks, design changes were suggested by the verifier which would remove the problems. These were evaluated for acceptability by the original designer. Finally a new design was verified incorporating many of the suggested changes. This demonstrated that the expected advantages were realized.

1 Introduction

Interactive proof is a very powerful method of ensuring the correctness of designs. However, in contrast to more automated formal verification methods, such as model checking, it is currently prohibitively time consuming. This is especially so when the idiosyncrasies of real circuits have to be considered. In this study, we investigated whether the verification task can be simplified by making appropriate design choices: that is whether a notion of *Design for Verifiability*, similar to that for testability, is of practical interest. The idea was originally introduced by Milne [4] in the context of automated state-exploration verification techniques. He only suggested very general ways that designs might be made more easily verifiable, such as using synchronous rather than asynchronous design, imposing the use of particular target architectures and providing language constructs that restrict the structures used.

Our study involved the verification of an existing hardware design. The original aim had been to verify the actual fabricated hardware. However, this was far more time-consuming than anticipated. Whilst conducting the verification we noted the factors that were causing the difficulties. It became apparent that particular aspects of the implementation were lengthening the verification time by significant amounts. Furthermore, it was clear that implementation changes could remove the problems. A criticism of making such changes might be that other design considerations such as speed or functionality would be sacrificed. The original designer therefore evaluated all the proposed changes for acceptability. In fact some of the changes suggested improve the design with respect to other such design goals. A new implementation incorporating the changes was then verified. However, as the verification study was conducted after the design had been fabricated none of the suggested changes have been incorporated into the actual in-use design.

The application considered was a real, fabricated component of a working Asynchronous Transfer Mode (ATM) network: the Fairisle 16 by 16 switching fabric [8]. It was designed and fabricated with no thought for formal methods. We attempted to verify the design using interactive, machine-checked formal proof with the HOL theorem proving system [6]. This built on our earlier work to verify the simpler Fairisle 4 by 4 switching fabric [1]. The latter has since been used as a verification case study for several other verification systems [10, 9, 3].

The original designer was an experienced hardware designer, who had much prior experience in the application area. The verifier, who suggested the changes, and produced the new design had no hardware design experience, though was a competent user of the HOL system.

In the remainder of this paper we first give an overview of the design considered and describe briefly the original verification attempt. We then describe those aspects of the implementation which caused most problems in the proof and

Figure 1: The cell format

suggest modifications that could overcome the problems. We discuss which changes were acceptable to the designer. We then describe our results verifying a new design which incorporates many of the changes. Finally we describe the general lessons learned and draw conclusions.

2 The 16 by 16 fabric

The design we considered is a *switching fabric*. Its function is to transfer data from a set of input lines to a set of output lines. The data arrives at each input in the form of a *cell*: a fixed number of bytes which arrive sequentially. Each cell starts with a *header* (see Figure 1). It includes control information which tells the fabric the output that that particular cell should be sent to. Cells are expected to arrive on all inputs in synchrony, both at the byte and cell level. An external, periodic *frame start* signal divides time into a series of rounds or *frames* in which cells arrive. Thus in a particular round the headers should arrive on the same clock cycle on all inputs submitting cells. However, the fabric does not know in advance which clock cycle it will be; just that all the inputs will be synchronised. The fabric determines the actual cycle using an *active bit* in the headers. When cell data is not being placed on the inputs, zeroed bytes will be input. The fabric therefore watches the inputs from the start of a frame until the active bit of any goes high. This indicates that the headers have arrived. In any particular frame, only one cell can be sent to a given output. However, it is possible that several input cells can be requesting a single output in one frame. In this case, the fabric must choose one cell and discard the others. This is done using priority information in the cell headers combined with round robin arbitration.

To summarise, the behaviour of the switching fabric is cyclic. In each frame, it waits for cells to arrive, reads them in, arbitrates any clashes, sends successful ones to the appropriate output ports, discards unsuccessful ones and sends acknowledgements back to the inputs.

The interface to the fabric is illustrated in Figure 2. The inputs and outputs of the fabric are connected to port controllers which are ultimately connected to network transmission lines. The port controllers append the headers to the cells, queue cells, store and retransmit those discarded by the fabric due to clashes, and perform other administrative

Figure 2: The fabric interface (showing one input and one output port connection)

tasks. Acknowledgement lines from the fabric inform the port controllers whether their cell was successful in each frame. The port controllers retransmit failed cells in a later frame.

Switching fabrics of differing sizes are implemented by a series of switching elements connected in a regular array. The simplest (4 by 4) switching fabric consists of a single element which connects 4 input ports to 4 output ports. It consists of an arbitration unit, a dataswitch and an acknowledgement unit (see Figure 3). Each of these modules is further subdivided into a hierarchy of simpler modules down to the logic gate level. The data switch performs the actual switching of cell bytes from inputs to outputs. The acknowledgement module generates the acknowledgement signals that are sent back to the input ports. Both these modules operate under the control of the arbitration unit. It generates internal timing signals based on the frame start input and the headers from the cell streams. It ignores the other cell data which is passed straight to the dataswitch. It first decodes the headers, does a preliminary filter of the cells based on their priority bits and then performs round robin arbitration for the remaining cells based on their requested destination. It outputs two control signals: *grant* and *outputDisable*. The former indicates which input ports are to be connected to which output ports in the current frame. The latter indicates when the grant signal is valid. When *grant* is not valid for a given output port zeros are output. This occurs for all ports from the start of each frame until the cells arrive, and over the whole frame for output ports that no cell has requested.

The 16 by 16 fabric is made from 8 elements in two stages connected in a delta network (see Figure 4). It consists of approximately 50 distinct (and independently verified) modules and uses a total of just under 4000 basic components (i.e. multiple input logic gates or single bit flip flops). Cells are initially input to the stage 1 elements. Each element makes an independent arbitration and routing decision based on the cell header information. Successful cells are forwarded to the appropriate stage 2 elements, where a further round of arbitration occurs based on a second set of routing information in the headers (see Figure 1). Only cells which are successful in both rounds are ultimately forwarded to the output port controllers. Acknowledgements from the stage 2 elements are passed back to the input port controllers by the stage 1 elements.

Although the elements of both stages perform basically the same task, they differ in several ways both from each other and from that of the 4 by 4 fabric. For example, the frame start signal is delayed on entry to the different elements by different amounts to adjust the internal timing. Also, the routing information for the two stages of the 16 by 16 fabric is read from different nibbles of the header.

Figure 3: The implementation of the 4 by 4 elements (numbers indicate line size)

Figure 4: The 16x16 switching fabric

Figure 5: Time taken to verify the 4x4 fabric modules

3 The verification

The formal verification was conducted using the HOL proof assistant: an LCF style theorem prover for higher-order logic [6]. A user of the HOL system gives proof commands in the form of Standard ML function calls. We followed the traditional approach to verifying hardware using higher-order logic [7, 5]. The verification consisted of proving for each module in the design, a correctness theorem of the form:

$$\vdash \text{Assumptions} \supset (\text{Implementation} \supset \text{Specification})$$

This states that the description of the implementation implies the specification under certain assumptions on the environment. *Implementation* gives a hierarchical structural description of the implementation down to the logic gate level: the components used, how they are wired together and which wires are inputs, outputs and local to the module. *Specification* gives the more abstract behavioural description consisting of the timing and functional behaviour of the module, mainly in the form of interval temporal style operators. *Assumptions* give the conditions on the environment (i.e. inputs) under which the module will satisfy the specification. The implementation, behavioural specification and assumptions are given in the form of higher-order logic relations on the inputs and outputs (as appropriate). Once the separate modules have been independently verified, their correctness theorems are hierarchically combined to give a correctness theorem for the whole device. In general, the proofs involve expanding definitions, performing case splits over different time intervals, and appealing to previously proved lemmas about the functional behaviour over the given intervals.

Initially, the 4 by 4 fabric was verified [1].¹ Several months were spent by the verifier understanding the implementation and writing formal specifications for each module as only limited informal documentation was initially available. Approximately six person weeks were then spent formally verifying the design. The breakdown of this time is given in Figure 5. It shows the cumulative time as each module was verified. The simpler modules, low in the design hierarchy, were generally verified before the higher, more complicated ones. Nearly half of the time was spent verifying just two modules: the top level of the arbitration unit which combines the priority decoding, arbiter and timing units; and the module which combines the arbitration unit, the dataswitch and the acknowledgement unit. They took longer partly because they were the most complex, requiring the most cases to be considered. A further reason was that the specifications contained errors – mainly over the details of the timing and over the form of the environment assumptions.

The plateau in the graph for the module DMUX2B4CA11 also corresponds to an error in the specifications. It arose because the verifier believed that the different bits from the control signal produced by the arbitration unit were read by the dataswitch on the same clock cycle when in fact they are read on consecutive cycles.

The proof of the 4 by 4 fabric was re-engineered for the elements of the 16 by 16 fabric [2]. These elements were verified much more quickly than the original, taking a matter of a few person-days each. This was because the new proofs could be obtained by modifying the existing proof rather than being created from scratch. However, it was clear that more time could have been saved if the implementation had been slightly different.

Next, the verification of the combination of elements to make the 16 by 16 fabric was attempted. The proof was completed up to virtually the last proof obligation. This obligation required that the stage 1 elements guarantee one of the environment assumptions of the stage 2 elements. However, the correctness theorem proved for the stage 1 elements was not strong enough to prove it. The problem does not mean the design is incorrect; rather, an extra environment assumption is needed for the stage 1 elements so that a stronger correctness theorem can be proved. Completing the verification with the new assumption would not be a significant effort in comparison to the full proof as it would involve simply modifying the existing proof. However it was felt that the time would be better spent looking at ways to prevent the problem arising in the first place.

Irrespective of the above problem, the uncompleted proof for the combination of the elements took far longer than any other part (weeks rather than days). Because of this it is especially important that attempts to speed the proof are targeted here. The problems were due largely to environment assumptions and timing details.

4 The suggested modifications

In this section we describe changes to the fabric that were suggested by the verification attempt. For each modification, we also overview any advantages and disadvantages.

4.1 Timing of frame start and cell arrival

One of the assumptions made by the 4 by 4 elements of their environment is that the active signal (i.e. the indication that cells have arrived) does not occur at particular times within a few cycles of the start of a frame. In the correctness statement for the fabricated back elements the assumption has the following form.

$$\forall t. (FrameStart\ t) \supset \sim (Active\ (t + 6) \wedge Active\ (t + 7))$$

We have simplified its presentation here for the purposes of exposition. It essentially states that if a frame starts at time t (i.e. $(FrameStart\ t)$), then the active signal should not be set at both $t+6$ (i.e. $(Active\ (t + 6))$) and $t+7$ (i.e. $(Active\ (t + 7))$).

This assumption was not documented. Its need was also not uncovered by the process of writing the formal specifications of the design or its modules. It was only when the machine-assisted proofs were attempted that its need was recognised. It arises because of the implementation of two modules in the arbitration unit: the timing unit which generates the main timing signal and the round robin arbiter. In both cases the assumption is needed because the module

¹ See URL <http://www.cl.cam.ac.uk/Research/HVG/atmproof/> for further details of the specifications and proofs, as well as other information about the project.

Figure 6: The effects of an assumption change on verification times.

would otherwise fail to reset itself consistently. This need can be removed by redesigning the modules. The only disadvantage is that a small amount of extra logic is needed. The exact timing varies between the different 4 by 4 element designs due to different delays placed on the frame start signal that adjusts their internal timing. The assumption appears in different forms in other modules.

In the current usage of the fabric, the frame size is 64 cycles and the active signal comes 8 cycles into the frame. When the fabric is used in this way the anomalous situation will never arise. It was thus not an issue considered by the designer. However, it has a significant effect on the verification as it resulted in extra proof obligations. When a module requiring the assumption was used, it was necessary to prove that it could be guaranteed by its environment: normally the output of some other module. Since the assumption is ultimately an assumption on the environment of the fabric such reasoning is required for modules all the way up the hierarchy.

A further way the assumption was time consuming was that the form used for the 4 by 4 fabric was too weak for the 16 by 16 fabric's elements. This was due to the extra delays placed on the frame start signal in the 16 by 16 elements. The assumption as stated could not be guaranteed on start up because of the unknown initial values in the registers delaying the frame start signal. This problem could only have been prevented if it had been known that the original 4 by 4 design was to be later modified for the 16 by 16 elements. Even then it is plausible that the initialisation problem would not have been noticed, given the need for the assumption itself was not initially realised.

The assumption change resulted in extra proof work modifying previously completed proofs. The resulting proofs were also slightly more complicated. This affected all modules whose correctness statement mentioned the assumption. It resulted in the stage 2 element taking 3 days to verify. The verification of a similarly modified module which did not need the assumption change took only a few hours (see Figure 6). Thus the assumption was directly responsible for nearly 3 days extra proof work for this reason alone.

It was also this assumption that ultimately caused the failure of the original verification. The stage 1 elements could not guarantee to the stage 2 elements that the active bits were zeroed. This was because the bits in question were not the same bits that the stage 1 elements needed to be zeroed for their own correct operation. The bits in question actually formed part of the data from the previous frame. An extra assumption that the last bytes of each frame do not hold data was needed.

Whilst the original design works perfectly well, the proof would have been greatly simplified if the need for the assumption had been removed. Time-wasting mistakes would also have been avoided. Furthermore the resulting design makes fewer assumptions about the environment and a potential initialisation anomaly is removed. It is thus much simpler. Any effort expended by the designer ensuring that other modules guarantee the assumption would not have been required.

4.2 Shared active and priority bits

The header byte of each cell is split into two halves. One half contains routing information for the stage 1 elements. It indicates which stage 2 element the cell should be directed to, the cell's priority and the active bit. The second half contains the corresponding information for the stage 2 elements (see Figure 1). This means that there are two active bits, which leads to an extra assumption that both are set together. However, there is no reason for one to be set without the other as it would just result in the cell being discarded.

Extra proof work is required, manipulating and using this assumption. It could be removed if the same active bit were used by both stage 1 and stage 2 elements. This would involve minor re-wiring. The duplicated priority information could be treated similarly. Making this change would free up extra bits in the header for other purposes. The change would also reduce the possibility of inconsistency due to errors in the port controllers. Finally, it would make the stage 1 and stage 2 element implementations more similar, reducing the time required to modify the proof for one design to apply to the other.

4.3 A fixed cell arrival time

The design allows cells, and thus headers, to arrive at any time within a frame provided that they all arrive together. This leads to relatively complex assumptions about the structure of a frame (i.e. cycle in which cells are processed). These assumptions change form for different modules and different element designs. Furthermore extra proof effort is required to ensure that the frame structure is preserved on the outputs of the stage 1 elements to satisfy the similar assumptions made by the stage 2 elements. Finally, this aspect of the design leads to the need for extra assumptions to ensure that the cells do not arrive close to the frame start (see Section 4.1).

An alternative would be to have the headers arrive at a fixed, known time after the start of the frame. This would simplify the definition of what constitutes a frame and thus reduce the amount of work reasoning with it. It would simplify the timing circuitry and make it easier to verify. It might also allow the delay on the data path to be shortened as the data would no longer be held up whilst the active signal was manipulated. Consequently, less logic would be required. As the frame structure would no longer be derived from the data stream, the current proof obligation that the frame structure is preserved as cells pass through the fabric would not be needed.

This change makes the fabric less flexible. In practice this is not a problem as the port controllers must already agree on a fixed time to inject cells. However, it would no longer be possible to use a single version of the element in switches which required different delays. Originally the extra flexibility was included because the fabric was designed before the port controllers. It was thought useful as it was not clear what the necessary delays required by the port controllers would be. However, this feature caused much confusion in the implementation of both the port controllers and larger fabrics. In retrospect the designer thought it had been a bad decision. The extra weight of the argument for ease of verification might have pushed the decision in the other direction.

4.4 Single behaviour stage 1 element

The stage 1 elements of the 16 by 16 fabric combine two different behaviours. In one behaviour, cells are sent to the output they request. In the second, cells requesting outputs 0 and 1 are sent to outputs 2 and 3, respectively, and *vice versa*. The two behaviours are controlled by an extra input to each of the elements. In the 16 by 16 fabric, these inputs are given hardwired values. The top stage 1 elements are hardwired to one behaviour, whilst the bottom stage 1 elements are hardwired to the other. This was done because a single interchangeable board design was wanted for the four top elements and the four bottom elements (see Figure 7). However, the two boards need to be mirror images of each other to ensure that cells are sent to the correct outputs. This means that on the top board requests for outputs 0 and 1 must

Figure 7: The need for different behaviour on the top and bottom elements

stay on the board, whereas on the bottom board they must go off the board. Thus they are sent to different outputs of the elements, which are then wired appropriately so that the cells do ultimately end up at the requested output port.

The two mode behaviour added several days to the verification time. The verification proceeded by first verifying a simpler version (modified from the 4 by 4 fabric design) which had a fixed behaviour like the other elements. These specifications and proofs were then modified for the real design. It is very easy to write a specification for elements hardwired to one behaviour or another, for example, by adding a function that maps physical ports to logical ports. However, the implementation does contain additional logic over the simpler design, to read the flag input and switch behaviours. Thus extra work modifying the original proofs and verifying the additional circuitry is necessary. Furthermore, the implementation would be simpler and use less logic without the two-mode behaviour as the extra circuitry required for switching behaviours would not be needed. Conducting the proof in 2 stages helped make it more tractable. It also meant the extra time required to prove the actual stage 1 elements over a more straightforward design could be seen (Figure 8). Additional time is also required to verify the connection of the elements into a 16 by 16 fabric.

The designer preferred the original design as the inconvenience of using two symmetrical board designs, over interchangeable boards, was too great.

4.5 Copy the last byte of a cell

The elements ignore the last data byte of each frame, thus increasing the minimum delay between frame start signals for a given cell length. The problem is that the arbiters disable the outputs one cycle earlier than is desirable. This is necessary because of the way the dataswitch part of the element is implemented. On each clock cycle, to determine which output the current byte should be sent to, the dataswitch consults the two bits of the grant control signal produced by the arbiter. One of those bits is sampled on the cycle before it is used, but the other is sampled on the same cycle. This ultimately means that the grant signal for the last cycle cannot be used as its value changes between the bits being sampled. The problem can be removed by adding extra delays at suitable points so that all the grant bits are read at the same time.

Figure 8: Comparison of proof times for different stage 1 elements

This aspect of the implementation gave rise to additional complexity in the specification. Extra work was required determining the subtleties of the timing and the reasons the fabric worked correctly. Manipulating the more complex specifications during the proof also took extra time. Finally, the fact that the bits of the grant signal were read on different cycles was missed in the original specifications. This resulted in extra time being spent discovering and correcting the error, as noted in Section 3. The specifications are also much cleaner without the change.

4.6 Delay the frame start

The frame start signal to the stage 2 elements is delayed by more cycles than to the stage 1 elements. This is because the cells arrive at the first stage 5 cycles later than at the second, having had to first pass through stage 1. However, the extra delay added to the frame start in the fabricated implementation is only 3 cycles. This would lead to the ends of cells being lost due to the next cycle starting if it were not for the spacing between frame start signals being longer than the cell length. Instead empty bytes are lost. Whilst the design still works, this is a flaw that ought to be corrected. Had any effort been made to shrink the frame period to a minimum it would either have been discovered and corrected or become a critical error.

In fact an earlier fabricated version of the 16 by 16 fabric had included no extra delay to the frame start on the stage 2 elements. This error was discovered by the formal verification. It had also been discovered by the designer when the fabric was in use. The extra three cycle delay was introduced to fix this bug.

If the frame start signal to the stage 2 elements was delayed by a further 2 cycles, the delay through the stage 1 elements would be exactly compensated for. Less proof work would then be required to verify the connection of the elements, as the reason it works would be less complicated. The environment assumptions would be easier to discharge, and the behaviour of the device would be cleaner.

5 Verifying an Improved Design

A new 16 by 16 fabric was designed, implemented and formally verified (though not fabricated), based on the modifications outlined in the previous section. Our aim was to demonstrate that the suggested advantages would be realised.

The new design incorporated the following changes which were compatible, acceptable to the designer and did not introduce any significant loss of functionality. Their introduction was in no way problematic.

- The arbiter was redesigned to reset correctly if the cells and frame start occur close together.
- The timing unit was redesigned so that it does not ignore cells which arrive close to the frame start.
- The separate active (and priority) bits were merged.
- Internal delays were added so that the extra cell byte is not lost.
- An additional 2-cycle delay was added to the frame start to the second stage elements so that it corresponds to the delay through the first stage.

In addition the new design used single mode front elements (see Section 4.4). This change would not be acceptable to the designer in a fabricated design. It was made here for pragmatic reasons related to the verification case study: time was limited, the new design was not to be fabricated anyway and the time necessary to add the extra behaviour to the elements was known because it had been done before. Thus little would have been learned by adding the extra behaviour.

A further implementation change was made due to an anomaly which was only found during the verification of the new design. An extra 5-cycle delay was added to the control signals into the acknowledgement unit of the stage 1 elements. This was done to compensate for the extra 5-cycle delay made to the frame start of the stage 2 elements. It synchronises the acknowledgement units in the two designs so that they are enabled to send acknowledgements at the same time, rather than being out of phase by 5 cycles. If this change were not made, a cell could theoretically be thrown away without the knowledge of the port controllers. However, this would only occur in a situation where very short cells were being used. Since this is not the way the design is intended to be used, it is not a serious defect. However, the change removes complexity from the specification of the fabric and consequently from the verification. The acknowledgement signal would have three distinct phases of behaviour without the change, rather than just two with it. The additional phase corresponds to the period where one stage of elements is ignoring the acknowledgement signal of the other. The change alters the externally visible timing of the active signal. It may therefore require a change to the port controllers so that they do not look for the acknowledgement too soon.

The aim of creating and verifying a new design was to show that the suggested advantages were realised. This was so. The problematic parts of the proof were removed. The troublesome assumption about the cells not arriving close to the frame start has been removed from the correctness statement. The resulting specifications of each module are much simpler and clearer than in the original design. In particular a single generic definition of a frame has been used for all modules. Previously several different versions were needed reflecting the variations in the details of the timing for the different modules in the design. Various lemmas which previously needed to be proven about the different frame definitions or about the environment assumptions were no longer needed. No significant extra proof work appears to have resulted, though the new version of the arbitration and timing units required slightly longer proofs. For example the proof of the timing unit was split into two modules as for the new design this was conceptually simpler. The proof of the arbitration unit involved an additional case: that which had previously been covered trivially by the assumption. However, both these units occur low in the hierarchy and are consequently straightforward to prove. In comparison to the higher modules, the proof times involved were small.

Ideally, to demonstrate the reduced verification time we would compare the time taken to complete the new verification with that for the original design. However, this does not give a fair comparison. As we have previously demonstrated [2] having a proof of a design is an asset that speeds later proofs. Even if theorems are not actually reused but reproved from scratch, the verifier's knowledge would be such that the second proof would be easier. If a different verifier performed the proof (which was not an option in our study), the time taken would depend on the relative skills of the two verifiers. Also the specifications for the new design would be less likely to contain errors, yielding a speed up of verification times. Furthermore, we integrated the verification with the design process for the new implementation. This further distorts the results of any comparison of proof times.

However, as the alterations to the design have been suggested on the grounds of removing work that had to be performed in the original, it seems sufficient to have noted that these problems of the proof have in fact been removed

and not replaced by new problems. As the verification of the redefined modules does not require significant amounts of new proof effort over the corresponding original modules, the changes have been vindicated.

6 General Lessons

In this section we overview general ways suggested by our study that designers could simplify the verification task.

Keep the interfaces between modules simple. A common feature of the acceptable design changes is that they involve simplifying the environment assumptions of modules. This reduces the verification cost by removing or simplifying proof obligations and reducing the possibility of mistakes being made in those assumptions. Mistakes in assumptions can be very costly in time. This is because they may not be discovered until a late stage in the verification when other modules are being verified. Also such assumptions must contain sufficient information to verify one module whilst only containing information that can be guaranteed by another. In practice, this often leads to the situation where the assumption is first strengthened, then weakened, only to be strengthened again later as the correct balance is ascertained. In the interim much proof work is wasted. This situation was partly responsible for the fact that the verification of just two modules in the original 4 by 4 fabric took half of the total verification time. Unduly complex assumptions can also turn out to be in an inappropriate form when design modifications are made. This results in extra proof work to generalise them. This was responsible for several days extra proof effort in the verification of the elements of the 16 by 16 design as well as the failure to complete the original 16 by 16 verification.

Because the existence of environment assumptions is not design specific, examining and simplifying assumptions may be a general way of reducing verification time without adversely affecting functionality or performance. Furthermore, it is something that, at least to some degree, a designer can think about in advance of the verification attempt. Thus, whilst the rigours of performing the verification are likely to suggest additional changes, some problems could be prevented from arising in the first place. This would be especially so if formal specification were used.

Push complexity down to the lowest levels. The biggest savings in verification time are made where modules deep in the hierarchy depend on assumptions that are discharged by modules much higher up. This occurs with the frame start and cell arrival time assumption in the 16 by 16 fabric. In this situation the assumptions have to be dealt with in the verification of many different modules. They can also change form and increase in complexity as they pass up the hierarchy. The result of removing such assumptions is often that extra proof complexity is pushed down to the simplest modules at the base of the hierarchy where it is most easily managed. It is likely that such low level modules can be verified automatically against their specifications, using for example BDD and state exploration techniques. Thus the extra complexity is of no consequence in such modules. If this results in the module having very clean behaviour, higher level modules will be much easier to verify.

Start with a clean specification and stick to it. The specification of the modified design is very clean. If a designer were to write down a specification of the exact intended behaviour of the design before starting implementation, it is likely that something similar would be produced. Many of the verification problems arose because specific implementation decisions introduced minor deviations from the clean specification. For example, in the 16 by 16 fabric the problem with the timing of the frame start and cell arrival arose in this way. When the designer is working from an informal specification, such changes are easily overlooked or ignored. With a formal specification the changes to the specification that such decisions make become explicit, and are harder to ignore. This suggests that designs would be easier to verify if a top-down design process was followed which started from a formal specification and gradually refined it into a hierarchy of modules.

Keep related designs as similar as possible. Clearly proof effort can be reduced if the number and degree of modifications made when a design is re-engineered is kept to a minimum. Designs should be made generic from the outset where possible. It is also important that the verification team are aware of possible future changes so that genericity can be built in.

7 Conclusions

We have shown that different implementation choices can have a significant effect on the speed of verification using interactive proof. Such changes do not necessarily compromise other design goals. We have demonstrated that this is so on a real, fabricated hardware design. Although we have only considered a single design, the nature of the changes identified suggests that the results are more generally applicable. Their nature also suggests that the results are not restricted just to HOL, but are more widely applicable. Design choices are routinely made for testability. This study shows they can also usefully be made for provability. We investigated the *post facto* verification of an existing design. Although this suggested ways that a designer could prevent problems arising, further work is required to demonstrate the extent to which design for provability can be integrated into the design process itself.

Unlike with testing and automatic verification techniques, interactive proof requires an understanding of why the design works as it should. It is from the process of doing the proof that much of the increased confidence in the design comes rather than from a yes/no answer that the correctness theorem can be proved. The proof process not only requires such understanding, it provides a tool to achieve that understanding. As a consequence it can act as a design aid; improving designs by highlighting flaws and unduly complex parts and helping to suggest alternatives and simplifications.

Perhaps most importantly, designing with provability in mind encourages design simplicity. The designer may have to work a little harder to ease the verifier's task. However, the result is a much cleaner design. This is vitally important for safety-critical systems where formal proof techniques are most likely to be used.

Acknowledgements

This work was supported by EPSRC Grant GR/K10294. We are grateful for the support of Mike Gordon and the members of the systems and automated reasoning groups at Cambridge. Simon Moore gave valuable advice on the redesign of the fabric.

References

- [1] Curzon P. Experiences formally verifying a network component. In: Proceedings of the 9th Annual IEEE Conference on Computer Assurance, IEEE Press, 1994, pp 183–193.
- [2] Curzon P. Tracking design changes with formal machine-checked proof. The Computer Journal, 1995; 38(2):91–100.
- [3] Garcez EHA. The verification of a ATM switching fabric using the HSIS tool. Technical Report WSI-95-13, Tuebingen University, 1995.
- [4] Milne GJ. Design for verifiability. In: Leeser M, Brown G. (eds), Hardware Specification, Verification and Synthesis: Mathematical Aspects, Springer-Verlag, 1990, pp 1–13 (Lecture Notes in Computer Science No. 408).
- [5] Gordon MJC. Why higher-order logic is a good formalism for specifying and verifying hardware. In Milne G, Subrahmanyam PA (eds), Formal Aspects of VLSI Design. North-Holland, 1986.
- [6] Gordon MJC, Melham TF (eds). Introduction to HOL: a theorem proving environment for higher order logic. Cambridge University Press, 1993.
- [7] Hanna FK, Daeche N. Specification and verification of digital systems using higher-order predicate logic. IEE Proceedings—Computers and Digital Techniques, September 1986; 133(E-5):242–254.
- [8] Leslie IM, McAuley DR. Fairisle: An ATM network for the local area. In: Proceedings of the SIGCOMM Conference on Communications Architectures and Protocols, Computer Communication Review, ACM Press, September 1991, 19(4):327–336.

- [9] Schneider K, Kropf T. Verifying hardware correctness by combining theorem proving and model checking. In: Alves-Foss J (ed), International Workshop on Higher Order Logic Theorem Proving and Its Applications: B-Track: Short Presentations, 1995, pp 89–104.
- [10] Tahar S, Curzon P. A comparison of MDG and HOL for hardware verification. In: von Wright J, Grundy J, and Harrison J (eds), Theorem Proving in Higher Order Logics: 9th International Conference, Springer-Verlag, 1996 (Lecture Notes in Computer Science No. 1125).