EXPERIENCES FORMALLY VERIFYING A NETWORK COMPONENT

In the
Proceedings of the 9th Annual IEEE Conference on Computer Assurance,
IEEE Press, 1994.

Paul Curzon
University of Cambridge Computer Laboratory,
New Museums Site, Pembroke Street,
Cambridge CB2 3QG United Kingdom
Email: pc@cl.cam.ac.uk

Errors in network components can have disastrous effects so it is important that all aspects of the design are correct. We describe our experiences formally verifying an implementation of an Asynchronous Transfer Mode (ATM) network switching fabric using the HOL90 theorem proving system. The design has been fabricated and is in use in the Cambridge Fairisle Network. It was designed and implemented with no consideration for formal specification or verification. This case study gives an indication of the difficulties in formally verifying real designs. We discuss the time spent on the verification. This was comparable to the time spent designing and testing the fabric. We also describe the problems encountered and the errors discovered.

1 Introduction

Communication networks are rapidly becoming all pervasive. Systems are increasingly being networked in the local area with applications using non-local services. In the wide area, telecommunications companies are turning to digital networks. As networks become all-pervasive, the consequences of errors in the design or implementation of network components become increasingly important. This is especially so if networks are used in safety-critical applications where communication problems could cause loss of life. For example a telephone network problem can contribute to loss of life if the emergency services cannot be contacted. Errors could cause the network to deadlock, particular links to crash, the service to be degraded to an unacceptable level, or even the whole network to crash. Network problems affect a wide range of users and applications and can cause whole systems or companies to grind to a halt [16, 17].

Asynchronous Transfer Mode (ATM) is a relatively new technology that is being adopted by the computer and telecommunication industries in local and wide area networks in response to changing communication demands. It is likely to be the most important transfer mode of the foreseeable future. It is being touted as a technology that can be used "everywhere": in wide-area, metropolitan area, local area and even desk area networks [14]. ATM systems could become high-volume products for which dependability is paramount. It is thus an important application for verification research.

We describe our experiences in formally verifying an ATM network component. This work is part of a larger project to investigate the formal verification of the Fairisle ATM network [13]. It is an experimental network, designed and built at the University of Cambridge Computer Laboratory. The network carries real user data and thus provides a realistic case study for the investigation of the formal verification of ATM network hardware. The network component we have verified is the Fairisle 4 by 4 switching fabric.

The 4 by 4 switching fabric forms the heart of the Fairisle switch. It performs the actual switching of data cells from input ports to output ports and arbitrates cell clashes. The formal verification work has been performed on completed implementations. This is generally considered

to be harder than integrating the formal specification and verification into the design process. This problem was exacerbated further since there was little documentation. A significant amount of reverse engineering was required. The behavioural specifications were largely deduced by examining the implementation.

We produced formal descriptions of both the implementation and its behaviour. We then used formal logic to rigorously prove that the behaviour prescribed by the description of the implementation satisfies the specified behaviour. In contrast to validation using non-exhaustive testing, the results hold for all valid sets of inputs, not just for some small subset. Formal verification corresponds in this sense to exhaustive testing. However, exhaustive testing is infeasible for all but very small designs due to the large number of possible input values. Formal verification is feasible because of the use of mathematics (such as induction) to consider the behaviour resulting from ranges of input values together.

The proofs described were carried out using the HOL90 theorem prover: a Standard ML implementation of the HOL system [6]. It is a machine implementation of a classical higher-order logic. It provides mechanical assistance to the proof process, ensuring mistakes are not made. The system will only call something a theorem if it has been rigorously proved.

There has been much work in the area of formal hardware verification, most notably in the area of microprocessor verification [1, 11]. There has been some previous work on the formal verification of network components. For example, Herbert initially used LCM-LSM and later HOL to formally verify an ECL chip: a local area network interface used as part of the Cambridge Fast Ring [9, 8]. It is of a similar complexity to our switching fabric, though the ECL proof took much longer to perform. This is an indication of the increased maturity of both HOL and hardware verification in general. Melham also used HOL to verify the T-ring: a very simple ring communication network that was designed as a formal verification case study [15]. Josephs *et al* produced a hand proof that switching elements similar to those proposed for the INMOS Transputer could be connected together in a regular way to implement a router of arbitrary size [10].

We do not go into details of the formal specifications and proofs here. We concentrate on our experiences performing the verification. The time taken was comparable to the time originally spent designing, implementing and informally testing the fabric. No errors were found in the fabricated implementation. This was unsurprising as the fabric had been in service for some time prior to the start of the formal verification work. Undocumented "features" were found, however, and many errors were discovered in the formal specifications written for the verification. This was also unsurprising since documentation of the design and its implementation was sparse.

The full details of the specifications are given in a 100 page literate document [3] derived from the HOL source files using the HOL **mweb** tool. An overview is given separately [4].

2 The Fairisle Switching Fabric

The Fairisle switch consists of three types of component: input port controllers, output port controllers and a switching fabric. The port controllers process incoming or outgoing cells of data. They manage the routeing tables, setting up and breaking down virtual circuits. Each is connected to either an input or output link of the switch, and to the fabric. The fabric switches cells from input port controllers to output ones. This is illustrated in Figure 1. The fabric is the place where cells contend for bandwidth. If different port controllers inject cells destined for the same output port controller into the fabric at the same time, then only one will succeed. The others will be rejected and must retry later. The fabric consists of a series of identical switching elements connected in a regular array. The simplest fabric consists of a single element. It is this fabric that we have formally verified. We have *not* verified the port controllers.

The port controllers append to each cell a *routeing tag*. The fabric uses this to determine which

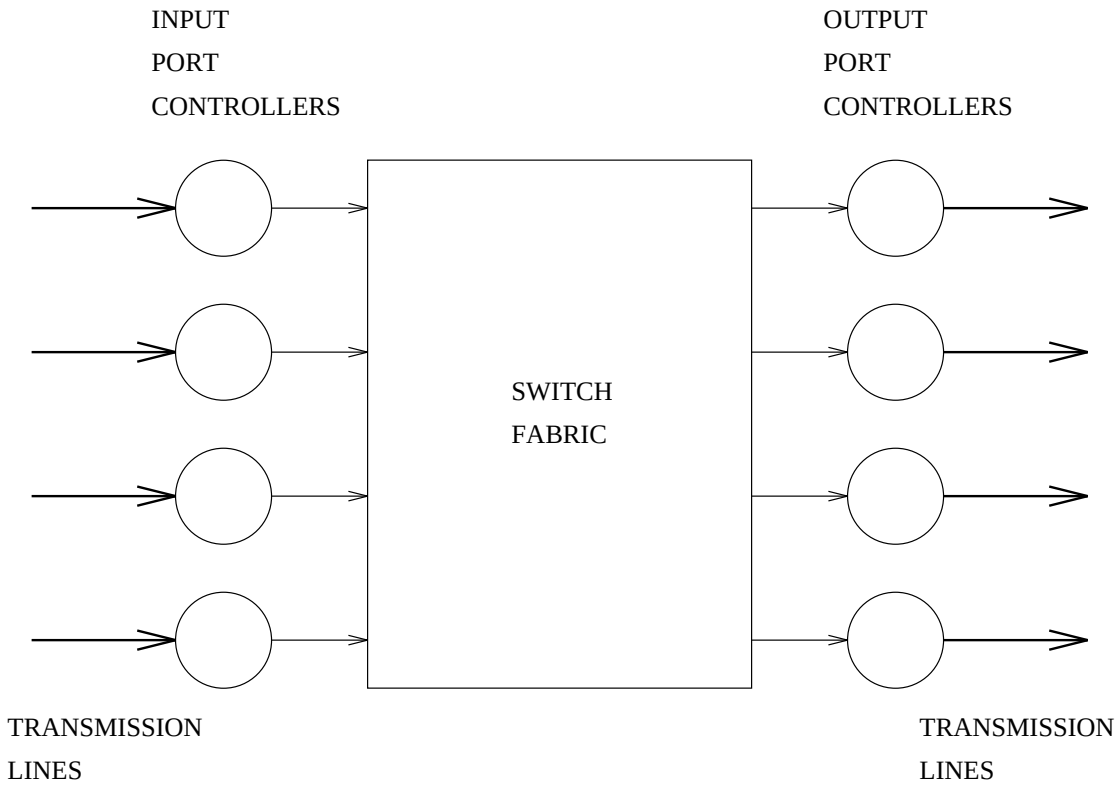

Figure 1: The Fairisle Switch

outgoing transmission link the cell should be transmitted on. It also includes one bit of priority information which is used by the fabric when arbitrating clashes. Arbitration takes place in two stages. Firstly, high priority cells are given precedence over low priority ones. Of the remaining cells, the choice is made on a round-robin basis. The input port controllers are informed of whether their cell was successful using acknowledgement lines. The fabric sends a negative acknowledgement to the unsuccessful input ports, but passes the acknowledgement from the requested output port to the successful input ports. This means the output port controllers may reject cells even if they successfully pass through the fabric.

The port controllers and fabric all use the same clock so bytes are read in on each link synchronously. They also use a higher level cell frame clock—the *frame start* signal. It ensures that the port controllers inject data cells into the fabric synchronously so that the routeing bytes arrive at the same time. The behaviour of the fabric is cyclic. In each cycle or *frame*, the fabric waits for cells to arrive, reads them in, processes them, sends successful ones to the appropriate output ports and sends acknowledgements. It then waits for the next round of cells to arrive. The boundaries of separate frames are determined by the frame start signal. Whenever it goes high, a new frame commences. When cells are not being offered by the input ports, they must inject zeros into at least the first bit of each byte. This is the active bit of the cell header. When a new frame starts, the fabric watches the active bit of each input port. The cells from all the input ports start when the active bit of any one of them goes high. The fabric does not know when this will happen. However, all the input port controllers must start sending cells at the same time within the frame, since any which have not set the active bit at the header time are assumed not to be transmitting cells for the whole of the frame. If no input port raises the active bit throughout the frame then the frame is *inactive*—no cells are processed. Otherwise it is *active*.

3 The Implementation

The 4 by 4 fabric is implemented on a 4200 gate-equivalent Xilinx programmable gate array. It was designed using a Hardware Description Language: Qudos HDL [5]. This is a simple HDL that allows the structure of hardware to be specified. It does not allow behaviour to be specified directly.

To formally verify the fabric, we needed a structural description of the implementation in a language with a formally defined semantics. Without this no formal reasoning about the behaviour of the circuit is possible. Unfortunately, no formal semantics exists for Qudos HDL. As we intended to perform the verification using the HOL theorem proving system, we ultimately needed a semantics in higher-order logic. We therefore manually translated the original descriptions to HOL-HDL. This is a subset of the HOL logic with the flavour of a hardware description language. It allows descriptions to be given which are very similar to descriptions in Qudos HDL. The translation could have been done completely mechanically, involving only the changing of syntax. However, for several reasons we decided to make changes to the description. It was not intended that these changes alter the design, only the description of it. Both descriptions should describe the same collection of logic gates. To be completely certain that changes to the design were not inadvertently made, the netlists from the two descriptions could be compared. Two kinds of changes were made: adding extra layers to the hierarchy and simplifying the description using features of HOL-HDL which were not available in Qudos HDL.

The designers of the switch expressed a desire for the new description to make use of features of HOL-HDL that were not available in Qudos HDL. The most notable was that Qudos HDL cannot describe words of words. The data input and output of the fabric consist of 4 byte-wide lines. They are thus best described as a word of length 4 with each field a word of length 8. This could be done in HOL-HDL, but in Qudos HDL it had to be described as 32 individual signals. The

Qudos HDL description was thus more unwieldily than necessary. Where the use of words allowed descriptions to be simplified, this was done. HOL-HDL also has a facility for describing generic hardware such as an n-bit adder implemented in terms of n 1-bit adders. The value of n can be a variable in the description. This is not possible in Qudos HDL. Such generic descriptions were used to replace several separate Qudos HDL descriptions. The changes made the descriptions clearer and made formal reasoning about the design more tractable.

Many of the modules in the original description were large and encapsulated several distinct tasks. We therefore added several levels of hierarchy that were not used in the original Qudos description. This made the description clearer. As the behavioural specifications were largely obtained by reverse engineering the HDL, this was very useful. It also facilitated the formal verification. For example, in the original description the top level module described the fabric in terms of input and output buffers, latches, a header decoder, priority filter, timing unit, arbiter, dataswitch and acknowledgement unit. We grouped all but the latches and buffers into a single unit at the top level. It was then subdivided at the next level into the arbitration unit, acknowledgement unit and dataswitch, with the arbitration unit further subdivided into a header decoder, priority filter, timing unit and arbiter. The resulting hierarchy is shown in Figure 2.

Both Qudos HDL and HOL-HDL have facilities for duplicating components. The different copies of a duplicated component can be wired in more general ways using HOL-HDL, however. In particular, arithmetic can be used to specify which bit of a word is connected to an input or output of a component. For example, we can specify that for all i, the $2i$-th bit of an output is connected to the i-th bit of a subcomponent. This means that the duplication construct could be used in the HOL-HDL version whereas the copies had to be written out in full in the Qudos version.

To illustrate the kinds of changes made, we will consider the Qudos HDL and HOL-HDL structural descriptions of a multiplexor component of the dataswitch – `DMUX4T2`. We first give the Qudos HDL definition.

```
DEF DMUX4T2(d[0..3],x:IN;dOut[0..1]:IO);
xBar:IO;
BEGIN
Clb:=XiCLBMAP5i2o(d[0..1],x,d[2..3],dOut[0..1]);

InvX:=XiINV(x,xBar);
B[0]:=AO(d[0],xBar,d[1],x,dOut[0]);
B[1]:=AO(d[2],xBar,d[3],x,dOut[1]);
END;
```

The description starts with declarations. The first line states that we are defining the module `DMUX4T2` and that it has two inputs: `d` which is 4 bits long (with bit positions numbered from 0 to 3) and `x` which is one bit. It has one two bit output, `dOut`. The second line declares a local variable, `xBar`. We then have the description of the layout, enclosed by `BEGIN` and `END`. The first statement is a dummy statement that provides information about the way the design should be mapped onto a Xilinx gate array. It provides no semantic information. The next statement describes a Xilinx inverter `XiINV`. It has input `x` and output `xBar`. This particular inverter is given the name `InvX` which is used by the simulator. There then follows two AND-OR logic gates, `AO`. They each produce one bit of the `dOut` output, using differing bits from the `d`, `x` and `xBar` signals. They are given the array name `B`, each being one entry. The individual bit positions are given in square brackets.

This description can be mimicked in HOL-HDL.

```
DMUX4T2((d,x),dOut)=
LOCAL xBar.
 XiINV(x,xBar) ∧
```

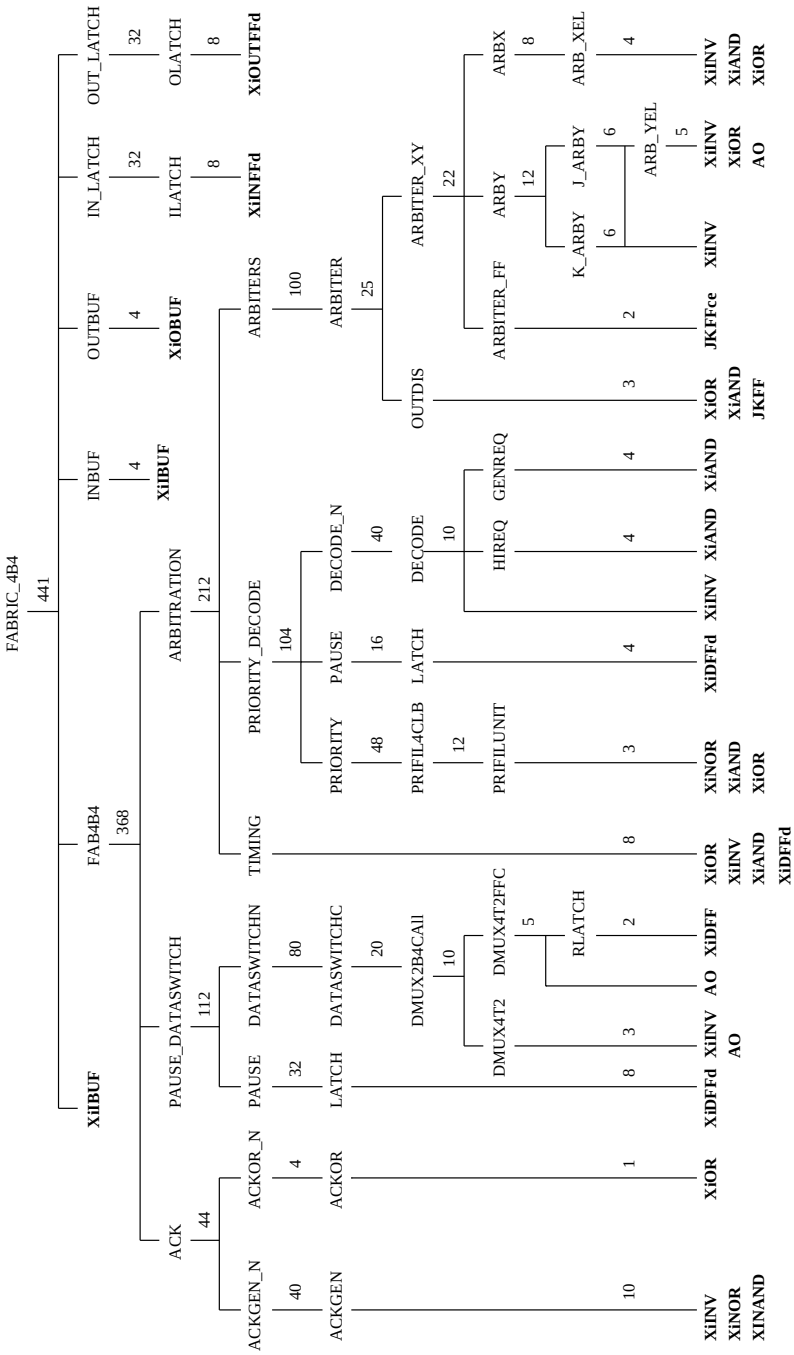

Figure 2: The design hierarchy showing the total number of primitive components in each module

```
 AO((SBIT 0 d,xBar,SBIT 1 d,x),SBIT 0 dOut) ∧
 AO((SBIT 2 d,xBar,SBIT 3 d,x),SBIT 1 dOut))
```

Definitions of modules take a single pair argument. The inputs form the first part of this pair and the outputs the second. Multiple inputs and outputs are grouped into a tuple in the appropriate part of the pair. We omit the information about the word lengths of the inputs and outputs. This is specified when the module is used (and in the correctness theorem). This is more flexible. The local variable `xBar` is introduced using the `LOCAL` construct. The three components are then given, separated by the ∧ operator. The dummy definition of the Qudos HDL is omitted as are the names of the components. They provide no semantic information and are not needed to perform formal verification. The bit positions are indicated using the function `SBIT`, and the inputs and outputs have the structure described above, but otherwise the component descriptions are the same. The unwieldiness of the HOL-HDL syntax could be overcome using parser and pretty-printer support.

In HOL-HDL, we can do better than the above. `d` can be thought of as being two signals each two bits wide. The input `x` chooses either the first bits of each of these signals or the second. We cannot describe this in Qudos HDL as structured signals are not supported, but we can in HOL-HDL.

```
DMUX4T2((d,x),dOut) =
LOCAL xBar.
 XiINV(x,xBar) ∧
 AO((SBIT 0 (SBIT 0 d),xBar,SBIT 1 (SBIT 0 d),x), SBIT 0 dOut) ∧
 AO((SBIT 0 (SBIT 1 d),xBar,SBIT 1 (SBIT 1 d), x), SBIT 1 dOut))
```

Now `d` is a two level signal. The low level bits are accessed using two calls to `SBIT`. The use of the two level signal makes the description closer to the designers mental model. Furthermore, whilst making the structural specification look superficially more complex, it simplifies the behavioural specification. This makes that specification easier to understand and also simplifies the verification task. We can now simplify the structural specification further. The two `AO` gates are performing identical functions, the only difference is in the bit positions. We can therefore introduce the duplication binder `FOR`. It introduces an index variable `i`, to range over the different values of the bit positions, from zero up to, but not including, the value after the keyword `TO`.

```
DMUX4T2((d,x),dOut) =
LOCAL xBar.
 XiINV(x,xBar) ∧
 FOR i ::  TO 2 .
  AO((SBIT 0 (SBIT i d),xBar,SBIT 1 (SBIT i d), x), SBIT i dOut)
```

The body of the duplication binder `FOR` describes the replicated components in terms of the index, here `i`.

4 The Formal Behavioural Specification

To formally verify a hardware design, a formal behavioural specification is needed. Ideally the formal specification would be written first and the device implemented from it. The fabric had been designed and implemented without any form of formal behavioural specification being created. There was only sketchy, informal documentation. Therefore the formal specification was reverse-engineered from the HDL description. This was a very time-consuming process and the early versions of the specification contained many errors. These are discussed later. To a large extent the formal verification process was used to aid the reverse engineering. Initially an “educated

guess" at the specification was made. As the proof progressed it was gradually corrected as the implementation was analysed and a better understanding of the design was obtained.

In addition to a behavioural specification of the fabric as a whole, a behavioural specification of each module was also needed. These were produced in the same way in a bottom-up manner. The behaviour of the lower level modules was first determined. These were then used as aids to determining the behaviour of upper level modules and ultimately the whole design. Auxiliary definitions were shared between the different module specifications.

Behavioural specifications of all modules were written before any proofs were carried out. This was because when the project started some tools were not available in the version of HOL being used. They were available by the time the specifications were finished. It would have been better if each module had been verified as it was specified. This would have prevented errors in the specifications of the lower modules from propagating up to higher ones. Also, because several weeks elapsed between the specifications being written and being used, the details of why the specifications were correct had to be partially rediscovered.

The formal behavioural specification of the fabric is a formal description of the timing diagrams of the output signals. It describes the expected values of the signals at each time instance. The specification is a relation on the inputs and outputs of the fabric and on the state which records the last successful inputs. As the behaviour of the fabric is cyclic, the formal specification is based on the frame cycles. The behaviour of each output signal over the period of a frame is described in terms of the state and values on the inputs during that frame. Frames can either be inactive or active. During an inactive frame no cells arrive, so no arbitration need be performed. During an active one at least one input port injects a cell into the fabric. An inactive frame could be thought of as just a degenerate case of an active one. However, the specification is cleaner if the two cases are treated separately. For inactive frames, the behaviour over the whole frame can be treated uniformly. For an active frame, the behaviour is typically split into two parts: that up to some fixed time after the active signal arrives; and that from this point until the end of the frame. Thus there are several cases to consider in the formal proof for each output signal. The main aspect of the functional behaviour of the fabric is the arbitration process. We specified this process as a function on the cell headers. Given a set of headers, the function describes the arbitration decision that will be made. The new state and the values of the acknowledgement and data signals to each output port are defined in terms of this function.

5 Time taken

As we mentioned earlier the specifications for each of the 43 modules (both behavioural and structural) were written prior to any proof. This took between one and two man-months. No detailed breakdown of this time has been kept however. Much of the time was spent attempting to understand the design. The structural specifications were adapted directly from the Qudos HDL. The behavioural specifications were more difficult. The specifier had no previous knowledge of the design or of the Fairisle Network. Documentation was minimal. There was a good English overview of the intended function of the fabric. This also outlined the function of the major components. Whilst it gave a good introduction, it was not sufficient to construct an unambiguous behavioural specification of all the modules. The behavioural specifications were instead constructed by analysing the HDL. This was very time-consuming. A consequence is that the formal specification describes what the implementation does. This may not be what the designer intended it to do. Errors in the design may have become "features" of the specification. However, since the fabric was designed and in use prior to the formal specification and verification being carried out, this was unavoidable. It can be alleviated to some extent if the designers thoroughly examine the top level specification. Ideally, formal specifications should be written by the designers of the module.

This would also aid the design process.

The proof of each module in the design was independent of the implementations of all other modules, including sub-modules of the module in question. Only the specifications of sub-modules were used. The separate proofs were then combined to give proofs for the modules' actual implementations in terms of the implementations of their component parts.

Approximately two man-months were spent performing the verification. Of this one week was spent proving general purpose theorems about machine words and signals. These theorems will be of use in future projects. Approximately 3 weeks were spent verifying the upper modules of the arbitration unit, and a further week was spent on the top two modules of the switch. 3-4 days were spent combining the correctness theorems of the 43 modules to give a single correctness theorem for the whole circuit. The remaining time of just over two weeks was spent proving the correctness theorems for the 36 lower level units. These proofs were automated using tactics which were developed throughout the verification, though normally additional human effort was required to finish the proofs. The breakdown of the time is given in Figure 3. It shows the cumulative time taken as each module was verified. The time spent proving general word theorems mentioned above is not included. On the whole, the simpler modules were verified first. The latches and buffers were particularly easy as they all had very similar structures. The proofs of the upper-level modules were generally more time-consuming for several reasons: there were several intervals to consider; they gave the behaviour of several outputs; and those behaviours were defined in terms of complex notions.

Tactics developed for the early proofs were used in the later proofs. This speeded up the more difficult ones. The verifier had not previously performed a hardware verification and was unfamiliar with the word library, though was a competent HOL user. Proof times were consequently reduced as experience was gained.

Much of the effort was expended in understanding informally why the implementation was correct. This was hampered to some extent by the delay between writing the specifications for a module and doing the proof. This meant that much time was wasted re-understanding the specifications and working out how the implementations actually worked. It would have been much better to perform the proof of a module when it was specified had this been possible.

Errors in the specifications slowed progress. When an error was present in a module, some time was spent attempting to prove theorems that were untrue. When this was discovered, the error had then to be located. The manner in which the proofs failed usually gave a strong indication as to the location of the error. However, it was not always immediately clear whether the error was in the behavioural or structural specification of the module, or because a component module's specification was too weak. Determining which was so involved examining the specifications of other modules as well as the faulty one. The specification then needed to be corrected and the proof completed. The main errors were in the modules `DMUX2B4CAll`, and `ARBITRATION`, where there are corresponding plateaus in the graph. The original behavioural specification for the latter contained several minor errors and some more serious ones. It was largely rewritten during the verification. We discuss the errors found in the next section.

After the proof had been successfully completed the behavioural specification was reviewed. Major changes had been made to it during the course of the verification. Consequently, some aspects of the specification were overly complex. The specification was therefore simplified and the proof redone. The changes involved the major reworking of the proofs for the timing module, the upper level modules of the arbitration unit, and the upper level modules of the fabric itself. Due to the modular nature of the design and proof, only the modules which were affected needed to be reverified. It took a total of one month to complete the original proofs of these modules. It took less than three days to modify the specifications and redo the proofs. Re-doing the proof was quicker for several reasons. The proofs were split into a series of lemmas. Many lemmas were

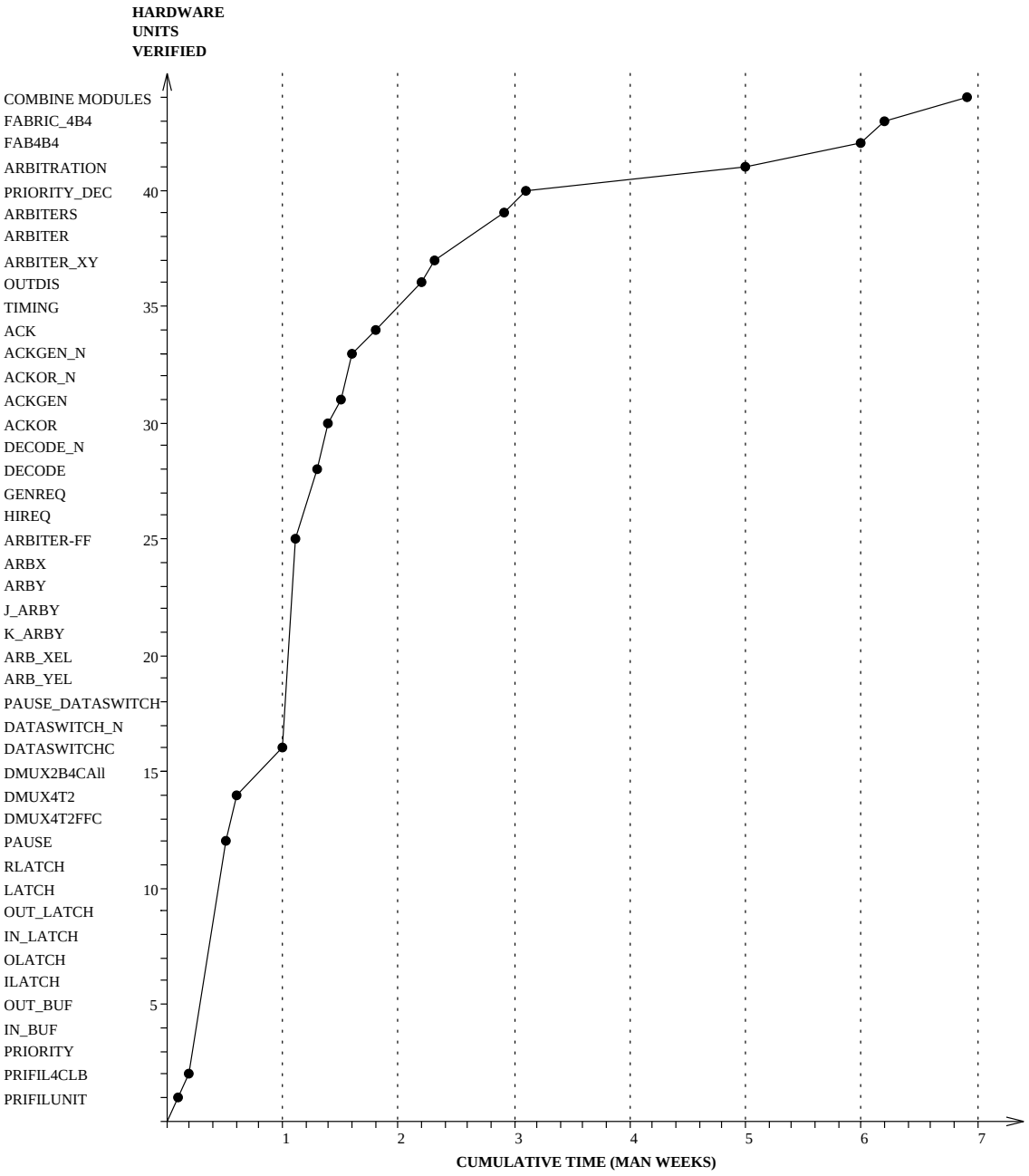

Figure 3: Time taken to verify the switching fabric

not affected by the changes and so their proofs did not need to be redone. The new specifications did not contain errors. The reason why the design was correct was well-understood because at an abstract level it was unchanged from the original. Most proofs that did need to be changed only needed to be changed in small ways. Thus the scripts could be rerun with only a few modifications. Some proofs were simplified by the changes.

No detailed record of the time spent designing and testing the fabric was recorded. The design evolved from earlier designs, and several different designs were produced at the same time, making it difficult to accurately estimate the time scale involved. The designer estimated that had it been designed from scratch the initial design time would have been in the order of several months. The time spent testing would have been in the order of several weeks. However, errors were discovered after the testing process had been completed when the fabric was in use. Thus the time spent to formally specify and verify the design was not unreasonable. Had it been performed as an integral part of the design, it is unlikely that it would have unduly slowed the design cycle. Furthermore, it is likely that the formal specification and verification would have been much quicker if done as the fabric was designed since much of the time was spent attempting to understand the design. Had formal verification been applied to the ancestors of the design, the formal verification could have tracked the changes made, with a minimal amount of time spent adapting the proof for the new generation. Similarly the proof could have been quickly adapted to the other versions of the fabric that were designed at the same time.

6 Errors Found During Formal Verification

In this section we give an overview of the errors discovered during the formal verification process in the various descriptions of the switching fabric. We outline the various kinds of errors which occurred. In the more interesting cases, we give an indication of why they occurred and how they were discovered. Our aim is to give a flavour of the wide range of errors that can occur in formal specifications as well as in implementations and how formal verification can help in their detection. All errors found during the course of the verification were corrected and the proofs completed successfully.

6.1 The Implementation

No errors were discovered in the actual implementation of the switching fabric. This is not surprising, since the fabric has been in use for some time. Also as noted above, the behavioural specification was written by examining the implementation. It is therefore possible that discrepancies between the implementation and designer's intended behaviour have become "features" of the behavioural specification.

6.2 The Structural Specification

Several errors were found in the `HOL` structural specification. These were introduced in the translation from Qudos HDL, due to the introduction of multi-level words, etc. For some errors this was due to the specifier misunderstanding the original description. In several places the wrong number of copies of a unit was specified. For example, in `DMUX4T2`, `(FOR i :: TO 1 ...)` was originally used to replicate the `AO` module. This created only one copy rather than the required two. This was because Qudos HDL takes an initial index and final index rather than the number of copies. This could have been avoided if a duplication construct had been defined in HOL to mirror the HDL more closely. This had originally been intended. The length was eventually used

since this made generic specifications simpler. In other modules, where a piece of hardware was duplicated using the length of one of the signals, the length of the wrong signal was used.

In several modules the sizes of the local signals were not specified. This information was needed in the proof. In `FAB4B4`, 2 bytes of a signal were selected when actually it should have been 2 bits from each byte.

In `DMUX2b4CAll` and `ARBITER`, two signals were incorrectly wired. This was discovered because the subgoal `([T, F] = [F, T])` was generated in the proof attempt. One side of this equality originated from the behavioural specification and one from the structural specification. This illustrates how the discovery of an error can give a strong indication of its cause. It was clear from the proof attempt that two signals had been swapped and also which signals they were from the context of the subgoal. It was not immediately clear in which specification they had been swapped.

6.3 The English Commentary

An English commentary was given with the behavioural specification of each module. Such a commentary is useful as it gives a brief, if not precise, overview of the behaviour of the module. This helps the verifier keep a mental picture of the purpose of each module. It would also be of use to engineers who are not familiar with the formal notation. An error was discovered in the English commentary of the `PRIORITY_DECODER`. It mistakenly stated that the priority decoder outputs one word per output port. It actually returns one word per input port. The formal specification was correct. The confusion arose because other components output this format. The error was discovered during the formal verification because the commentaries were being used to construct informal arguments to guide the formal proof.

6.4 The Behavioural Specifications

Many errors were found in the behavioural specifications of the modules. Most involved the incorrect specification of word lengths. Such errors were generally easy to detect and correct. The specification of the NOR gate was correct only for 2 input gates, but was used for gates of varying sizes. When originally specified it was thought that only 2 inputs would be used. This error was noted in the first proof in which an incorrect NOR gate occurred. One of the primitive word operators, `WSEG`, which selects a segment from a word, was incorrectly used. This was because the specifier had misunderstood its definition, assuming it took the end points of the segment as arguments, when it took one end point and a length. Again this was spotted in the first proof where it was used.

The timing of several modules was incorrectly specified. For example, in the specification for the module `TIMING`, an event was stated to occur at the same time as the frame start signal when it actually occurred on the subsequent cycle. Such errors were normally easy to detect and correct. Goals of the form `(ts = ts + 1)` were obtained. In fact, in the specification for the upper modules, the timing was purposely worked out in this way. Educated guesses were made about when events occurred and used in the specification. The actual values were then discovered during the proof and corrections made. This was easier than attempting to work out the timing by hand.

It was specified that the two bits of one of the signals to the dataswitch were sampled at the same time. In fact the implementation samples them at different times.

When initially writing the specifications it was assumed that the same definition of a frame between successive frame start signals could be used for all modules. However, the frame start signal is passed to all modules with no delay, whereas other signals suffer delays at various points in the circuit. In particular, the active signal is delayed at several points. This means that the

definition of a frame must vary between modules to account for the different relative times of the frame start and the active signal arriving at a module.

In several modules, the structure of the signals output was confused in a similar way to the error in the English commentary mentioned earlier.

Some of the specifications contained redundant information that either was not required for the proof, or was essentially stated twice within the specification. Whilst this did not affect the proofs for the modules in question, having unduly complex specifications would have complicated the proofs of the upper levels.

The specification of the arbiter did not give its behaviour on the last cycle of a frame in which no cells arrived. This was discovered because a subgoal had to be proved about the value of the grant signal at this time, but no information was available. The initial specifications for the upper modules in the hierarchy did not consider inactive frames as a special case. It was believed that inactive frames were covered in the behaviour given. However, this was not so. Consequentially an extra case needed to be added.

In several places the expression `SBIT k` which selects the k-th bit of a signal was used when what was needed was `($= k o BNVAL)` which converts the word to a number and then tests if it was equal to k. This arose due to confusion over the form in which the data was being stored on the outputs of those modules.

6.5 The Correctness Statement

An additional assumption needed to be added to the correctness statements of some modules. It was known about by the designers, though it did not appear in the informal documentation. It concerned the effect of the active signal arriving close to the frame start signal. It had initially been thought that the fabric would function correctly irrespective of when the active signal arrived. This was not so. An assumption was therefore needed that the active signal did not arrive at an inopportune moment. This assumption appeared in various forms for various modules as well as in the full correctness statement for the fabric. The fabric has no control over when these signals arrive as they are determined by the external environment. The design of the port controllers must ensure that the assumption is upheld. The assumption could have been included in the specification of the modules concerned rather than being explicit in the correctness theorem. This would have made little difference to the proofs.

7 Conclusions

We have demonstrated that a fully machine-checked formal verification of a real piece of communications hardware can be conducted in a time scale comparable to that required for its original design, implementation and informal testing. This was despite the formal specification and verification only starting after the design had been implemented; the documentation being sketchy, and the verifier having no previous knowledge of the design or its application. Whilst no errors were found in the implementation, problems were found in the original versions of the formal specifications. These were corrected and the verification completed. We also discovered and formally documented assumptions about the environment which must hold for the switching fabric to operate correctly. Whilst this information was known by the designers it was not documented. This could have led to errors if the fabric was used in a way other than that originally intended.

We have a rigorous description of the behaviour of the fabric, and of all its constituent modules. Having formally verified the implementation against it, we can have a high degree of confidence that the fabric does have that behaviour, provided it was correctly fabricated from the HDL description. This will be of use if changes are made to the design, and when interfacing the fabric to other

components of the switch. The proofs can quickly be modified for other designs of the switching element since correctness results for the modules can be reused if the modules are reused in the other designs [2].

We verified all modules down to the gate level. This was done to illustrate the feasibility of such a complete formal verification. However, the hierarchical methodology allows a more pragmatic approach to be taken if desired. Modules that are simple enough to be exhaustively simulated, or for which there is already a high degree of confidence for other reasons do not need to be formally verified. They can be taken as being basic modules, as was done with the specifications of the logic gates. Their formal specifications are then assumed to be correct. The formal verification of the rest can be carried out as normal. This allows more effort to be expended on the verification of modules where errors are thought most likely to occur. However, it should be remembered that not only must the implementation of the module be correct, but the formal specification must be an accurate description of it.

Many errors were found in the original formal specifications. This highlights that just like implementations, specifications are hard to get right. The main reasons for there being so many errors are that the specifier was not originally familiar with the designs being specified and because very little informal documentation was available. The specifications would probably have contained fewer errors if written by the designers during the design process, or if they had produced informal documentation for each module. Errors corresponding to those found in the behavioural specification could just as easily have been in the structural specification. The formal verification would have found such errors in the same way. The exercise does illustrate how well formal verification can discover errors and ensure that their correction does not introduce new errors, whether they are in the implementation or specification. Many errors concerned the sizes of words. These might have been discovered earlier if the sizes of the words could have been included in the type information – dependent typing. Then some of the errors might have been discovered during type-checking. Some systems such as VERITAS [7] and Nuprl [12] have this ability, though HOL does not.

Several errors arose due to the different representation of the request information in the headers of cells used by different modules. Initially the request information has the form of two bits per input port giving the binary version of the output port number. This information also appears as 4 bits per input port. Each bit is then a flag indicating whether a particular output port is making a request for an input. Elsewhere it is in a similar format except with 4 bits per output port with flags indicating whether an input port is making a request for the output. Furthermore, in the top level behavioural specification, ports are referred to by natural numbers. With more complete informal documentation of the implementation, such errors would have been less likely.

HOL-HDL could in principle be used in future projects instead of Qudos HDL to give the original structural descriptions of hardware. However, before this is sensible, more tools are required, for example, for simulating the HOL-HDL descriptions and for obtaining netlists from them. A quick way this could be achieved would be to write a translator from HOL-HDL to Qudos HDL. The Qudos tools could then be used. As the two languages are so similar, this would be relatively easy. The additional information needed, such as node names, could be provided using dummy definitions which semantically did not use the extra information.

Acknowledgements

This work was supported by SERC grant GR/J11133. I am grateful to Mike Gordon, Brian Graham, Ian Leslie, Wai Wong and the members of the Automated Reasoning Group in Cambridge for their help and advice.

References

[1] Avra Cohn. The notion of proof in hardware verification. *Journal of Automated Reasoning*, vol. 5, pp. 127–138, 1989.

[2] Paul Curzon. *Tracking design changes with formal verification: a network component case study.* Submitted for publication.

[3] Paul Curzon. *The formal verification of the Fairisle ATM switching element.* Technical Report 329, University of Cambridge Computer Laboratory, 1994.

[4] Paul Curzon. *The formal verification of the Fairisle ATM switching element: an overview.* Technical Report 328, University of Cambridge Computer Laboratory, 1994.

[5] K. Edgcombe. The Qudos quick chip user guide. Qudos Limited.

[6] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: a theorem proving environment for higher order logic.* Cambridge University Press, 1993.

[7] F.K. Hanna and N. Daeche. Dependent types and formal synthesis. In C.A.R. Hoare and M.J.C. Gordon, editors, *Mechanized Reasoning and Hardware Design*, pp. 49–68. Prentice Hall, 1992.

[8] J.M.J. Herbert. *Case study of the Cambridge fast ring ECL chip using HOL.* Technical Report 123, University of Cambridge, Computer Laboratory, February 1988.

[9] J.M.J. Herbert and M.J.C. Gordon. Formal hardware verification methodology and its application to a network interface chip. *IEE Proceedings Part E, Computers and Digital Techniques*, vol. 133, 1986. Special issue on Digital Design Verification.

[10] Mark B. Josephs, Rudolph H. Mak, Jan Tijmen Udding, Tom Verhoeff, and Jelo T. Yantchev. High level design of an asynchronous packet-routing chip. In J. Staunstrup and R. Sharp, editors, *Proceedings of the Second IFIP Workshop on Designing Correct Circuits*, pp. 261–274, January 1992.

[11] Warren A. Hunt Jr. *FM8501: A verified microprocessor.* Technical report, Institute for Computing Science, University of Texas at Austin, September 1985.

[12] M. Leeser. Using Nuprl for the verification and synthesis of hardware. In C.A.R. Hoare and M.J.C. Gordon, editors, *Mechanized Reasoning and Hardware Design*, pp. 49–68. Prentice Hall, 1992.

[13] I.M. Leslie and D.R. McAuley. Fairisle: An ATM network for the local area. *ACM Communication Review*, vol. 19, no. 4, September 1991.

[14] I.M. Leslie, D.R. McAuley, and D.L. Tennenhouse. ATM everywhere? *IEEE Network*, March 1993.

[15] T.F. Melham. *Higher Order Logic and Hardware Verification*, vol 31 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1993.

[16] P.G. Neumann. Inside risks: Some reflections on a telephone switching system. *Communications of the ACM*, vol. 33, no. 7, pp. 154, July 1990.

[17] Lauren R. Wiener. *Digital Woes: why we should not depend on software.* Addison Wesley, 1993.