Tool-Supported Dataflow Analysis of a Security-Critical Embedded Device

Chris Mills **Colin J. Fidge** **Diane Corney**

Faculty of Science and Technology,
Queensland University of Technology, Brisbane

Abstract

Defence organisations perform information security evaluations to confirm that electronic communications devices are safe to use in security-critical situations. Such evaluations include tracing all possible dataflow paths through the device, but this process is tedious and error-prone, so automated reachability analysis tools are needed to make security evaluations faster and more accurate. Previous research has produced a tool, SIFA, for dataflow analysis of basic digital circuitry, but it cannot analyse dataflow through microprocessors embedded within the circuit since this depends on the software they run. We have developed a static analysis tool that produces SIFA-compatible dataflow graphs from embedded microcontroller programs written in C. In this paper we present a case study which shows how this new capability supports combined hardware and software dataflow analyses of a security-critical communications device.

Keywords: Information security evaluation; Dataflow analysis; Static analysis; Embedded devices

1 Introduction

Security-critical communications devices used to safeguard data confidentiality and integrity in government, military and industrial applications must be rigorously evaluated before they are deployed. Typical 'domain separation' devices used to control the flow of information between classified and unclassified communications networks include data diodes (which enforce unidirectional information flow), encryption devices (which allow classified data to be sent over insecure networks), trusted filters (which constrict information flow) and keyboard-video-mouse switches (which allow a single workstation to access both high-security and low-security computers).

International standards, such as the *Common Criteria for Information Technology Security Evaluation* (ISO 2009), mandate information security, or 'infosec', evaluations of such devices. For instance, within Australia the Defence Signals Directorate follows such standards to produce a list of trustworthy devices, known as the *Evaluated Products List*[1].

A particularly challenging aspect of 'high-grade' infosec evaluations is to trace all (potential) dataflow paths through the device. With respect to the device's electronic circuitry this process is notoriously tedious and error-prone, but it becomes virtually impossible when embedded microprocessors are encountered on the circuit board. The number of dataflow paths through embedded program code far outweighs the number of physical connections in the surrounding circuitry and, unlike a circuitry schematic diagram, potential dataflow paths through software are not self-evident from mere inspection of the source code.

To help alleviate this problem we recently completed a static analyser (Fidge & Corney 2009) which can extract dataflow graphs from Embedded C programs in a form compatible with an existing tool for reachability analyses of digital circuitry (McComb & Wildman 2005). The combination of these two tools thus promises to support seamless automated analyses of dataflow through both the electronic circuitry and embedded software of security-critical communications devices.

In this paper we present a detailed case study demonstrating for the first time how these tools can be used together to analyse an actual domain separation device, tracing dataflow through both its hardware architecture and embedded software. The device itself is a testbed specifically intended for experimentation with infosec evaluation processes. The analysis produced all of the known dataflow paths through this device, as well as revealing some that were not anticipated.

2 Previous and related work

Overall our concern is with automated tools that can help an information security evaluator understand the (potential) flow of data through an electronic device, including both its electronic circuitry and embedded software.

There are, of course, numerous electronic circuit simulators available as both educational and debugging aids. These include Spice[2], the Electric VLSI Design System[3] and NGSpice[4]. However these are for modelling simple electronic components, semiconductors and logic gates, not microcontroller software.

This research was funded in part by the Defence Signals Directorate and the Australian Research Council via ARC Linkage-Projects Grant LP0776344.

[1] http://www.dsd.gov.au/infosec/epl/
[2] http://bwrc.eecs.berkeley.edu/classes/icbook/spice/
[3] http://www.staticfreesoft.com/
[4] http://ngspice.sourceforge.net/

There are also many multiprocessor simulators such as PTLSim[5] for the x86 microprocessor, CASPER[6] for the OpenSPARC T1, the SESC SuperESCalar Simulator[7], and the IBM Full Systems Simulator[8] for the PowerPC processor, but these tools generally focus on simulating a single processor at the level of individual instruction cycles.

Much closer to our needs are simulators for entire circuit boards, together with their embedded microprocessors, including commercial tools such as Wind River Simics[9] and OVPSim[10].

However, all of the above-cited tools are simulators for helping a developer debug a device by examining one functional behaviour at a time. An infosec evaluator is instead faced with the problem of analysing a given device which is presumed to be functionally correct and does not need debugging. Furthermore, a security evaluator needs to consider all possible behaviours of the device, not just a few. This requirement is best served not by a simulator but by a static analyser which can explore all of the device's behaviours at once. Finally, none of the tools cited above are designed specifically for security evaluations.

Much more useful for this purpose are tools that treat security-critical circuitry as a graph which can be analysed topologically. For instance, the Universal Virtual Laboratory includes a circuit analysis module which can determine whether or not two seemingly-different circuits are topologically equivalent (Mahalingam, Butz & Duarte 2005). More importantly, however, the Secure Information Flow Analyser, SIFA, performs topological analyses of circuitry schematics specifically to support information security evaluations (McComb & Wildman 2005).

We therefore used the SIFA tool as the starting point for our own research; its capabilities are described further in Section 3.1 below. In essence, the goal of our overall project is to extend SIFA with the ability to analyse embedded program code as well as circuitry.

3 Dataflow analysis tools used

Before presenting the case study, this section briefly describes both of the tools that were used, namely the Secure Information Flow Analyser (McComb & Wildman 2007) and our new C-to-SIFA Converter (Fidge & Corney 2009).

3.1 The Secure Information Flow Analyser

SIFA, the Secure Information Flow Analyser, is an open-source[11] software tool developed for the Defence Signals Directorate to assist with infosec evaluation of electronic circuits (McComb & Wildman 2005). It incorporates a simple graph editor to allow device models to be constructed manually, but can also import circuitry schematics expressed in the VHDL hardware design language.

SIFA represents electronic circuitry as a graph of *ports*, which form the basis for its reachability analyses (McComb & Wildman 2006). Typically ports denote physical pins and connections on a circuit board. Ports can be grouped to form *components*. These usually represent discrete electronic components on the board such as logic gates, integrated circuit chips, connectors, etc. SIFA allows components to be grouped hierarchically, thus providing a highly flexible modelling capability. Furthermore, SIFA treats all identically-named ports as denoting the same physical object. This allows circuitry diagrams to be split horizontally into different 'pages', with identically-named ports acting as off-page connectors, or vertically into layered models, allowing the same circuit to be described at different levels of abstraction simultaneously.

SIFA provides a variety of graph-theoretic functions for analysing models of security-critical circuitry (McComb & Wildman 2007). These include identifying all components between two points in the graph (which helps exclude components that have no security significance), finding cutsets between two points (which helps identify places in the circuit where infosec evaluations can be done most efficiently), and comparing two different graphs for overall equivalence (which allows an abstract model of expected data flow to be compared with the actual data flow in the concrete circuit).

However, SIFA's most important function is its ability to identify all dataflow paths between selected points in a graph, typically between a high-security data source and a low-security data sink. Since a circuitry graph is usually fully-connected (i.e., every electronic component is connected directly or indirectly to every other one), SIFA uses the concept of a device's operating *modes* to allow such graphs to be partitioned meaningfully. The user can define intra-component data flow with respect to particular modes. Modes are further divided into normal and 'fault' behaviours, with a probability attached to the latter. (SIFA has no semantic understanding of modes, however, using them merely as a syntactic way of partitioning the search space.)

SIFA thus performs a mode-specific analysis of inter-component reachability and presents the user with a list of those paths through the circuit that connect selected data sources and sinks in particular modes. The infosec evaluator can then inspect each such path to determine whether or not it poses a security risk. While adequate for circuits comprised of simple electronic components only, this process encounters difficulties coping with the complex behaviours of embedded microprocesors. The infosec evaluator is obliged to separately analyse the program code to determine how data may flow through these components.

3.2 The C-to-SIFA Converter

To solve this problem, we recently completed a 'C-to-SIFA Converter'. This is a compiler-like program that converts Embedded C code to SIFA-compatible dataflow graphs capable of being integrated into hardware circuitry models. Its input consists of computer programs written in Custom Computer Services' C dialect for Programmable Integrated Circuit microcontrollers[12], and its output is an XML description of a dataflow graph that can be loaded directly into SIFA. A preliminary description of the principles underlying the tool can be found elsewhere (Fidge & Corney 2009), with a more detailed description of the final implementation to appear in a forthcoming paper.

To model (potential) data flow through program code the tool uses the *Augmented Static Single*

[5] http://www.ptlsim.org/
[6] http://coe.uncc.edu/∼kdatta/casper/casper.php
[7] http://iacoma.cs.uiuc.edu/∼paulsack/sescdoc/
[8] http://www.research.ibm.com/systemssim/
[9] http://www.windriver.com/products/simics/
[10] http://www.ovpworld.org/technology_ovpsim.php
[11] http://sifa.sourceforge.net/
[12] http://www.ccsinfo.com/

```
if (u > 0) {
    t = t + v;
} else {
    t = w;
}
```

Figure 1: Example of a conditional statement.

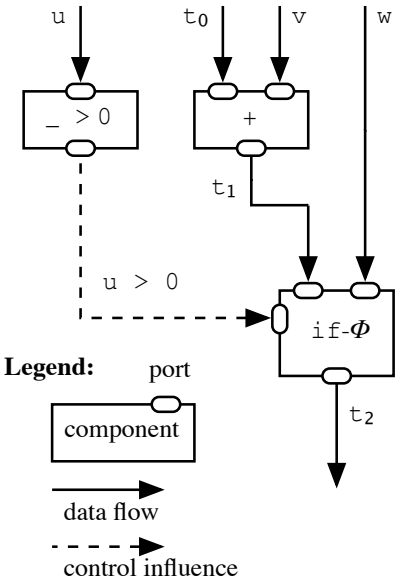

Figure 2: Dataflow graph generated by the C-to-Sifa Converter for the code fragment in Figure 1.

Assignment representation, originally developed for performing 'taint analyses' of security-critical programs (Scholz, Zhang & Cifuentes 2008). In particular, this representation considers not just explicit data flow between program variables, but also the implicit information flow created by one variable's value exercising control over assignments to another variable (Sabelfeld & Myers 2003). For instance, given the C program fragment in Figure 1, the C-to-Sifa Converter will produce the Augmented SSA dataflow graph in Figure 2. As in traditional dataflow graphs (Cytron, Ferrante, Rosen, Wegman & Zadeck 1989) it uses a 'ϕ' node to merge alternative dataflow paths through the `if` statement, in this case showing that variable `t`'s final value (t_2) may be derived either from the initial values of variables `t` and `v` or from variable `w`. In addition, however, the Augmented SSA graph also shows relevant control flows, in the same way as Gated Single Assignment representation (Ballance, Maccabe & Ottenstein 1990), in this case showing that variable `u`'s value exercises control over the final value of variable `t`.

Apart from implementing the basic conversion from imperative programming code to dataflow graphs, we also needed to extend the analysis to handle program constructs peculiar to embedded code. These included input and output statements that interact directly with the surrounding hardware, low-level, non-block structured control-flow statements such as **break**s and **continue**s, asynchronous control flow via hardware interrupts, and byte- and bit-level data operations.

The case study described below also highlighted some practical issues that needed to be solved within the C-to-Sifa Converter. For instance, since the device analysed contains two separate microcontrollers it was necessary to uniquely distinguish the dataflow nodes generated for each of the two embedded programs (because Sifa unifies all identically-named nodes) via a command-line option for prefixing the names of graph nodes with a microcontroller-specific identifer. Also, the large number of dataflow nodes generated for program code relative to its surrounding circuitry makes it difficult to interpret the long dataflow paths generated by Sifa, so the source code program's line number is included in the name of each dataflow graph node generated.

Most significantly, the user needs a way to link the microcontroller pins appearing in the circuitry diagram to corresponding input and output statements in the program code (Fidge & Corney 2009). A variety of potential solutions to this were contemplated, such as adding a 'bridging' component to the Sifa model to explicitly link hardware and software features, or providing a configuration file to the C-to-Sifa Converter to tell it what names are used for the microcontroller's pins in the hardware schematic. For the purposes of this particular case study, however, it was found to be expedient to simply directly edit the pin names in the (hand-crafted) hardware model to match those in the (automatically-generated) software model, especially since only a handful of the many pins on the microprocessor chips were used to transfer data.

4 The case study

To test the combined capabilities of Sifa and the C-to-Sifa Converter we performed a small, but complete, case study to show how potential data flow can be traced through both the hardware and software of an embedded domain-separation device.

4.1 The data diode device

The subject of the trial was a 'data diode device' produced by Australia's Defence Signals Directorate[13] as an unclassified and non-proprietary testbed for experimenting with infosec evaluation techniques (Mallen 2003). Our project team was given access to the device's design drawings, circuitry schematics and code listings, as well as a functional version of the device itself. A data diode device is typically used as part of a gateway between a high-security network and a low-security one, in order to ensure that there is no information leakage from the former to the latter.

The particular data diode device analysed here (Mallen 2003) contains two circuit boards connected by a ribbon cable as shown in Figure 3. The 'red' circuit board is connected to the high-security network via an RS232 serial cable and the 'black' circuit board is similarly connected to the low-security network. (This split architecture is intended to aid security evaluation of the device; high-security data should be found on the red circuit board only and the ribbon cable forms a narrow, well-defined bottleneck between the two security domains.) Both circuit boards contain their own Programmable Integrated Circuit microcontroller each running a different program written in Custom Computer Services' C dialect. Both microcontrollers directly control LEDs on the device's front panel to display its communication status ('ready for data' or 'waiting for acknowledgement'). Two switches on the front panel ('reset' and 'ack mode') are connected directly to the black microcontroller, and indirectly through the ribbon cable to

[13] http://www.dsd.gov.au/

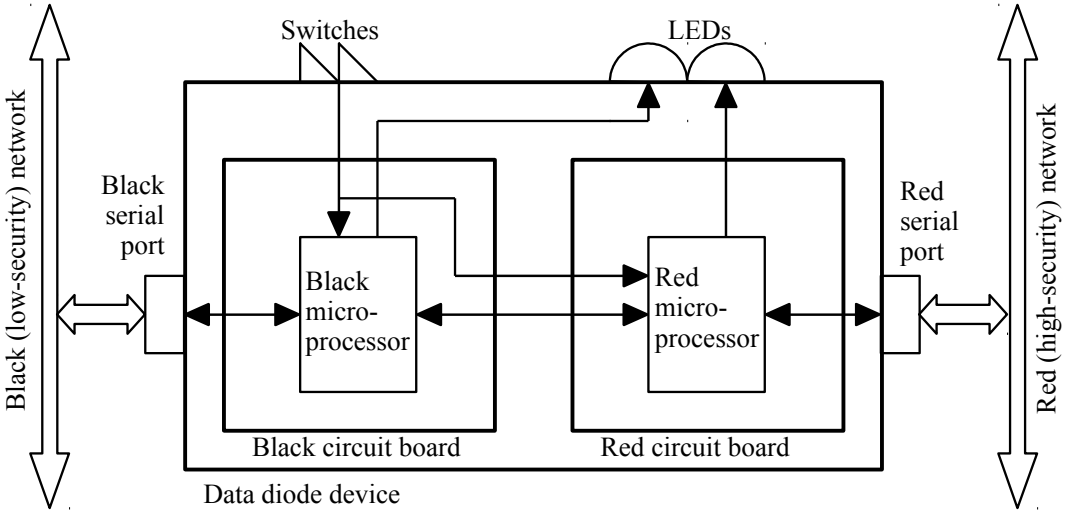

Figure 3: Block architecture of the data diode device.

the red microcontroller, to allow the operator to control the device's operating mode.

The data diode device's primary function is to allow data bytes to flow from the low-security network to the high-security one (i.e., from left to right in Figure 3) but not vice versa. However, to support communication over unreliable networks, this particular device also allows acknowledgements to be returned from the high-security network to the low-security one (i.e., from right to left in Figure 3). Such a capability is, of course, clearly dangerous because it allows information to flow from the high-security domain to the low-security one.

To (partially) mitigate this threat, the acknowledgement function is directly controlled by the operator via a front panel switch. Furthermore, entire bytes returned by the high-security network are not directly forwarded to the low-security one. Instead, the red microprocessor compares the returned byte with the one just sent. Depending on whether or not they match it sets one of two binary signals sent to the black microprocessor. Finally, the black microprocessor converts these signals into one of two characters ('`Y`' or '`N`') returned to the low-security network, thus constricting (but not entirely eliminating) the flow of information in the unsafe direction.

Overall, therefore, this data diode device offers an ideal testbed for infosec evaluation procedures since it has both a well-defined safe behaviour (the black-to-red data path) and a potentially unsafe behaviour (the red-to-black acknowledgement path).

4.2 Modelling and analysis process for the case study

To perform the analysis a model of the data diode device's hardware layout was first developed using SIFA's built-in editor, as shown in Figure 4. No VHDL representation of the circuitry was available, so the model was constructed manually, but this was not a major problem since this device's hardware is relatively simple; the black circuit board's model contained nine distinct components and the red board's model contained eight. (As is usual in these evaluations, power circuitry components, such as capacitors and resistors, were not modelled.) Appropriate mode-specific connectivity through each of these components, except for the microcontrollers, was defined for the two main operating modes of the data diode device, namely 'ack mode on' and 'ack mode off'. (Another advantage of the data diode device as a testbed is that its significant operating modes are obvious in its design.)

Next, the source code programs for the two microcontrollers were processed by the C-to-SIFA Converter. (Both programs are written in the Embedded C dialect for the PIC16F877 microcontrollers used in the data diode device.) Although the programs being analysed were quite small, the resulting dataflow graphs were still highly complex. The 'black' program consisted of only 106 lines of commented, formatted C code, plus a 248 line header file, but resulted in a graph containing 195 SIFA ports grouped to form 87 dataflow graph components. Similarly, the red program's 109 lines, plus header file, generated 200 ports forming 89 dataflow components. Part of these graphs is shown in Figure 5. (SIFA does not have an in-built graph layout tool, and the C-to-SIFA Converter merely generates nodes in a simple grid, without giving consideration to layout issues such as minimising line cross-overs. Fortunately, the infosec evaluator will not normally be obliged to study these graphs, relying merely on the output from the analysis, unless an exceptionally-detailed understanding of a particular dataflow path is required.)

It was then possible to load both the hardware and software models into SIFA, select source and sink nodes, and automatically analyse the model to find dataflow paths of potential security signficance. A variety of analyses were performed to ensure that all the dataflow pathways anticipated for this device were detected by the combined hardware-software model. Several of the paths returned by SIFA were then hand-checked in order to ensure that they conformed with our understanding of the way the device processes and forwards data. Doing this confirmed that the entire toolchain was working correctly and also helped us understand some unexpected, but logically 'correct', false-positive paths produced. (Inevitably a static analysis such as that performed by SIFA will to some extent overapproximate the actual paths that occur dynamically. While we can seek to minimise such false-positives, their existence is a fundamental limitation of static analyses.)

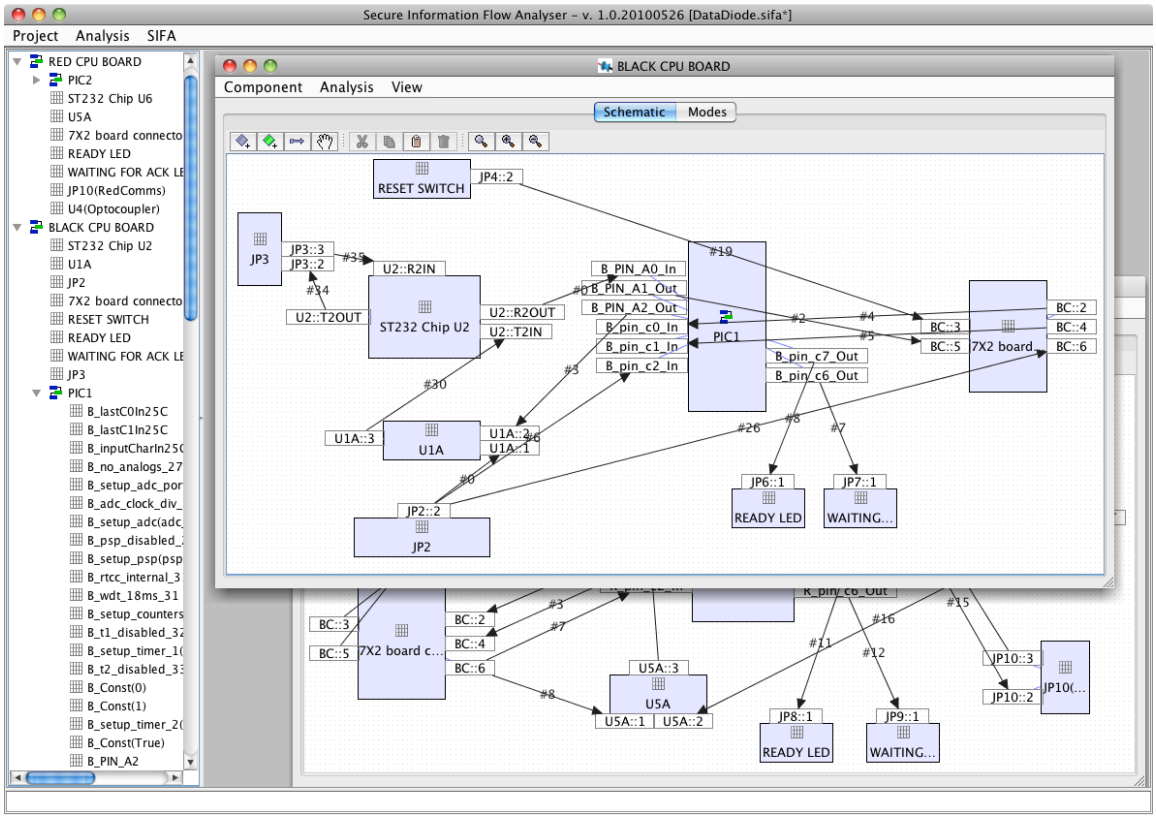

Figure 4: Models of the data diode's black and red circuit boards in SIFA's editing window.

5 Dataflow analysis results

Having completed the hardware and software models a number of dataflow analyses were performed to test the combined capabilities of the existing SIFA tool and our new C-to-SIFA Converter.

5.1 An explicit dataflow path

As an initial test we selected the incoming line of the data diode device's black serial connector as the data source of interest and the outgoing line of the red serial connector as the data sink, in order to identify (safe) dataflow paths from the low-security network to the high-security network via the data diode. As expected, SIFA reported the existence of one such path. This path, which comprised 20 distinct steps, represents the 'normal' flow of data bytes through the device.

Such paths are essentially just a list of ports, but to make them easier to interpret SIFA's interactive interface allows the user to single-step through the trace, automatically highlighting corresponding components in the graph and the operating modes in which they can be traversed. Doing this for the path found in this case allowed us to see how data can travel from the black network to the red one, via both hardware and software within the data diode device, and relate this path back to the original circuitry schematics and code listings (Figures 6 to 9).

In this case, starting from the black serial port (on the left of Figure 6), data bytes travel via the RS232 receiver component to pin A0 of the black microcontroller. The microcontroller's program (Figure 7) reads these bytes into a local variable, `inputChar`, and later sends them to pin A1. (The pins operated on by the `getc` and `putc` statements are determined by the preceding '`#use`' compiler directive.) This is

```
66    if (getNextChar==TRUE) {
          ...
70        #use rs232(baud=9600,Xmit=PIN_A1,
              Rcv=PIN_A0,parity=n,bits=8)
71        inputChar = getc();
72        // Disable ready LED
73        output_low(PIN_C7);
75        if (input(PIN_c2)) {
76            // Set waiting for ack LED
77            output_high(PIN_C6);
78            lastC0 = input(PIN_C0);
79            lastC1 = input(PIN_C1);
80        }
81        putc(inputChar);
82    }
```

Figure 7: Program code (lines 71 and 81) that transfers data from black microprocessor pin A0 to pin A1.

an example of direct data flow between hardware pins and software variables via explicit assignments in the program code, and demonstrates the C-to-SIFA Converter's ability to model these relationships.

From black microcontroller pin A1 the bytes travel to the red circuit board via the ribbon cable (right-hand side of Figure 6). On the red circuit board (left of Figure 8) the bytes travel via an optocoupler (used to ensure unidirectional data flow along this circuit) and enter the red microcontroller via its A0 pin.

Similarly to the other embedded program, the red microcontroller's code (Figure 9) transfers data bytes between its hardware pins A0 and A1 via an intermediate software variable, `rxChar`. From pin A1 each byte is forwarded to the data diode device's red serial port via an RS232 driver (right-hand side of Figure 8).

SIFA's identification of this expected data path

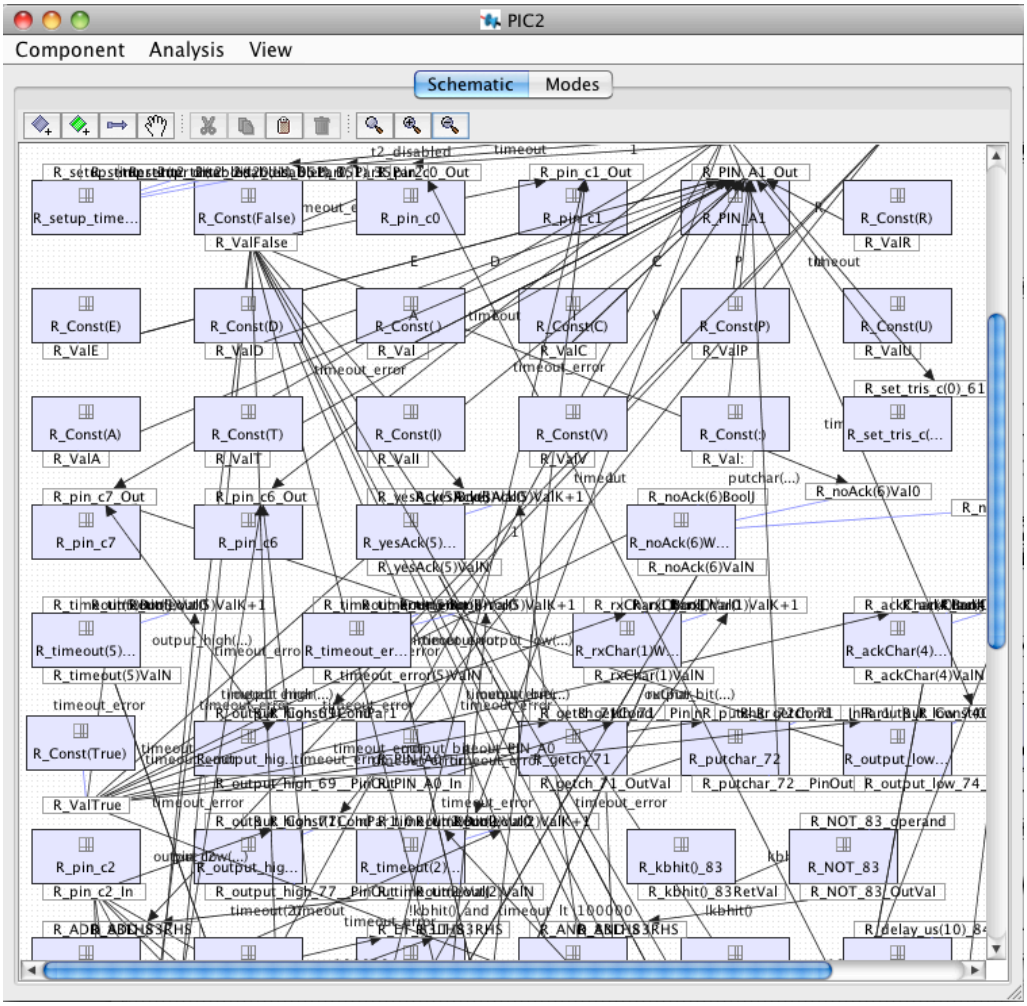

Figure 5: Part of the automatically-generated (and unformatted) model of data flow through the red microprocessor's software in Sifa's editing window.

```
66    while (TRUE)
67    {
69        output_high(PIN_C7);
70        #use rs232(baud=9600,Xmit=PIN_A1,
              Rcv=PIN_A0,parity=n,bits=8)
71        rxChar = getc();
72        putc(rxChar);
74        output_low(PIN_C7);
75        if (input(PIN_C2)) {
76            ...
```

Figure 9: Program code (lines 71 and 72) that transfers data from red microprocessor pin A0 to pin A1.

helped confirm the correct functioning of the C-to-Sifa Converter and demonstrates the ability to find composite hardware-software dataflow paths created by explicit data assignments.

5.2 Some implicit dataflow paths

More importantly, we then analysed the model using the data diode's red serial connector as the data source and its black connector as the sink, in order to identify potentially unsafe data flows from the high-security domain to the low-security one. Given the obvious dangers associated with the data diode device's acknowledgement function it was no surprise that Sifa identified the existence of such a path, but it was interesting to note that ten distinct high-to-low paths were produced, the longest of which involved 46 steps from source to sink.

Upon investigation, it was discovered that this large number of dataflow paths in the 'reverse' direction is due to the numerous conditional ('**if**') statements in the part of the program code that processes acknowledgements. For instance, both microcontrollers have program code that is conditional on the position of the 'ack mode' switch on the data diode device's front panel. Also, the red microcontroller executes different code depending on whether or not the byte returned by the high-security network is the same as the last byte sent to it. Similarly, the black microcontroller's program tests the values of both the positive and negative acknowledgement signals generated by the red microprocessor and performs different actions accordingly. Putting all of these alternative behaviours together accounts for the many distinct dataflow paths found by Sifa. Furthermore, the computational complexity involved in traversing these paths accounts for their significant length.

For instance, one of the potentially dangerous dataflow paths from the high-security domain to the low-security one concerns negative acknowledgements, produced when the data diode sends a byte to the high-security network but a non-matching byte is returned. Sifa's analysis shows that this path starts from the red serial port (on the right of Figure 10)

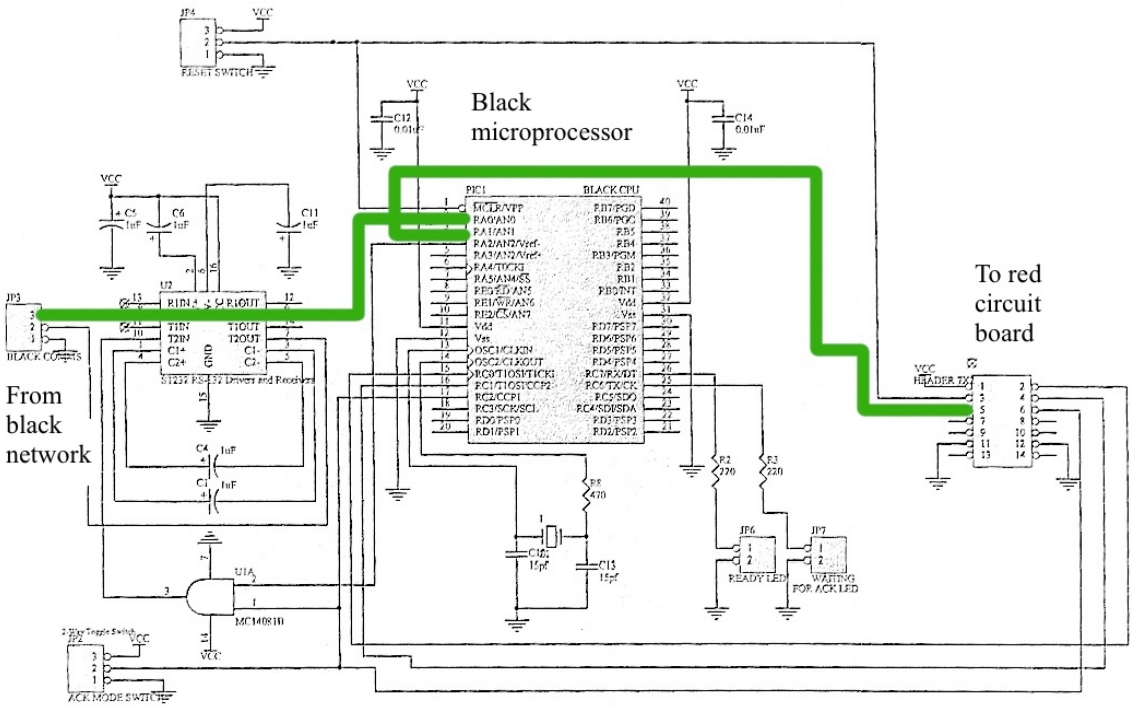

Figure 6: Data path (left to right) through the black circuit board via the black microprocessor.

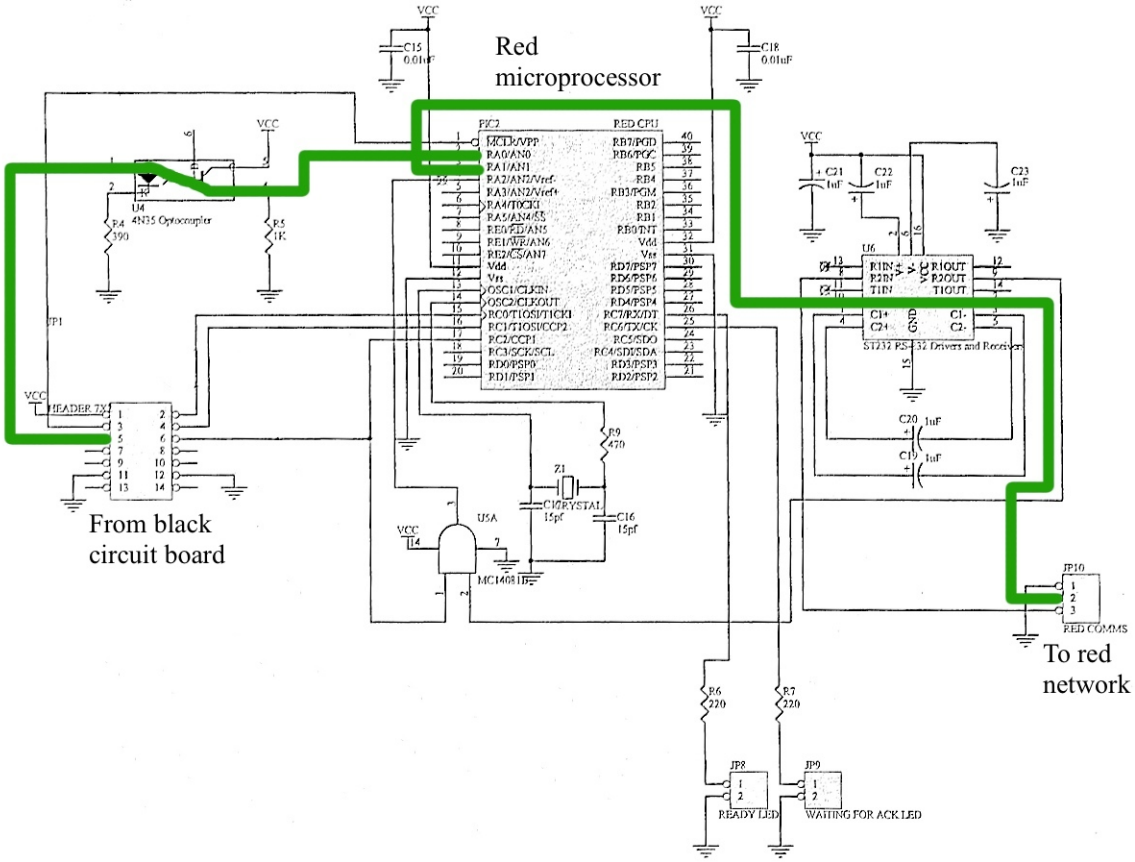

Figure 8: Data path (left to right) through the red circuit board via the red microprocessor.

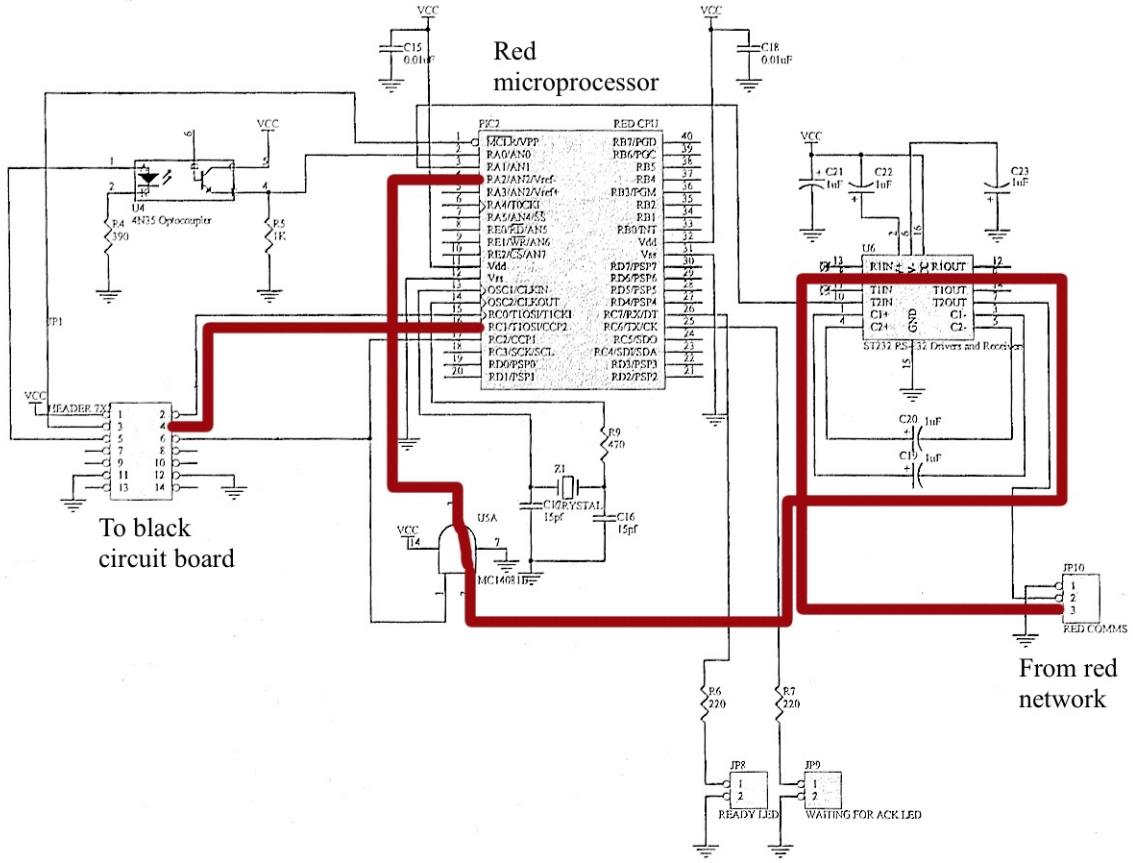

Figure 10: Control path (right to left) for negative acknowledgements through the red circuit board via the red microprocessor.

and travels through the red processor board's RS232 receiver and an AND gate before reaching pin A2 on the red microcontroller. The AND gate (bottom centre of Figure 10) is connected to the 'ack mode' switch on the data diode device's front panel (via the black circuit board) and is used to ensure that acknowledgement data reaches the red microcontroller only when the data diode device is in acknowledgement mode.

Sifa's trace through the red microcontroller's program code for this particular path (Figure 11) shows that the byte is read from hardware pin A2 into software variable `ackChar` (line 88). Later this variable is compared to the last byte sent to the high-security domain (line 99), held in variable `rxChar`. If the values do *not* match then the binary signal produced by microcontroller pin C1 is toggled to indicate a negative acknowledgement (line 104).

Notice in this code that there is no direct transfer of data from pin A2 to pin C1. The byte received via pin A2 *influences* the binary signal sent via pin C1, but no values from the byte are forwarded directly. This is, therefore, an example of implicit information flow between software variables and hardware pins, again confirming the C-to-Sifa Converter's ability to capture such system properties.

From red microcontroller pin C1 the signal then travels directly via the ribbon cable (left of Figure 10) to pin C1 of the black microcontroller (from the right in Figure 12).

The black microcontroller's program code repeatedly samples the signal on this pin to see if it has changed, in which case it sends a negative acknowledgement character 'N' to the low-security network. These multiple samples account for some of the different dataflow paths found by Sifa. For instance, a

```
78    #use rs232(baud=9600,Xmit=PIN_A1,
         Rcv=PIN_A2,parity=n,bits=8)
87    if (kbhit()) {
88       ackChar = getc();
89    } else {
90       timeout_error = TRUE;
91    }
93    output_low(PIN_C6);
95    if (timeout_error) {
96       noAck = !noAck;
97       output_bit(PIN_C1, noAck);
98    } else {
99       if (ackChar == rxChar) {
100         yesAck = !yesAck;
101         output_bit(PIN_C0, yesAck);
102      } else {
103         noAck = !noAck;
104         output_bit(PIN_C1, noAck);
105      }
106   }
```

Figure 11: Data-flow path (lines 88, 99 and 104) through the red microprocessor's code that translates data received from pin A2 into a 'negative acknowledgement' control signal sent via pin C1.

short path through the black microcontroller's code (Figure 13) occurs when the signal sampled from pin C1 (line 92) is used in a condition which directly controls whether or not the 'N' character is sent (line 95).

However, since the sampled signal is compared with a previous sample from the same pin, a longer

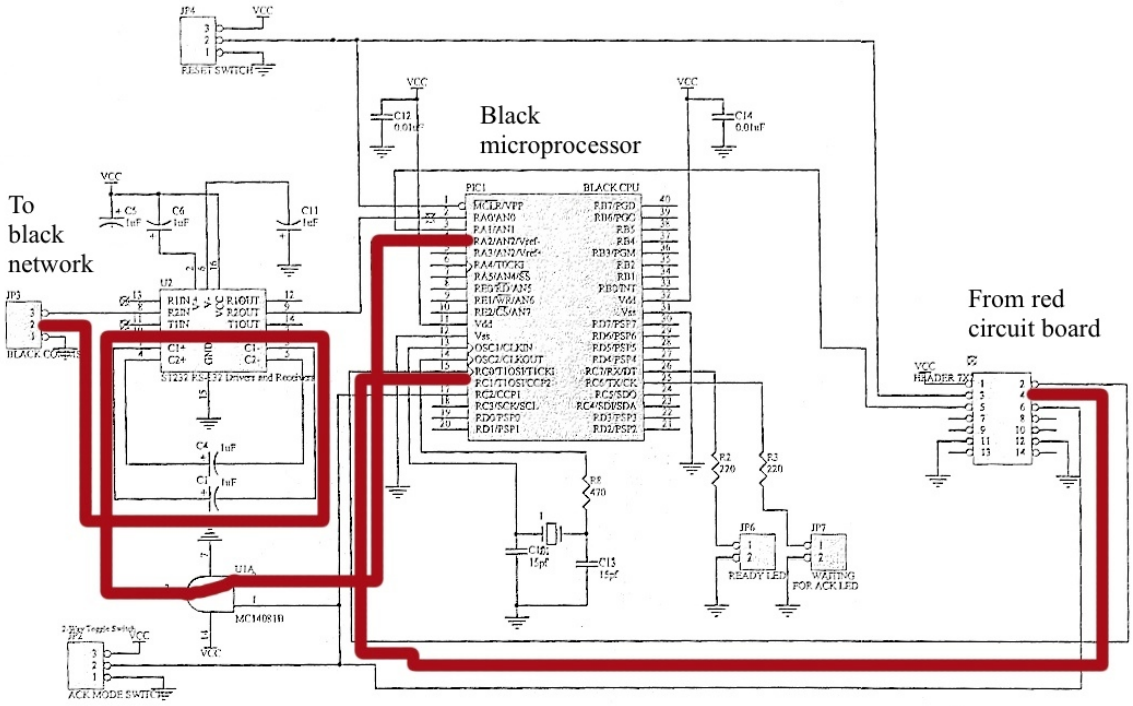

Figure 12: Control path (right to left) for negative acknowledgements through the black circuit board via the black microprocessor.

```
75        if (input(PIN_c2)) {
77           output_high(PIN_C6);
78           lastC0 = input(PIN_C0);
79           lastC1 = input(PIN_C1);
80        }
81     ...
84     if (input(PIN_C2)) {
85        if (input(PIN_C0) != lastC0) {
86           ...
91        }
92        if (input(PIN_C1) != lastC1) {
93           if (!getNextChar) {
94              #use rs232(baud=9600,
                Xmit=PIN_A2,
                Rcv=PIN_A0,parity=n,bits=8)
95              putc('N');
96              getNextChar = TRUE;
97              output_low(PIN_C6);
98           }
99           lastC1 = input(PIN_C1);
100       }
101    }
```

Figure 13: Two data-flow paths (lines 92 (left) and 95, and lines 79, 92 (right) and 95) through the black microprocessor's code that translate a control signal received from pin C1 into a 'negative acknowledgement' data value sent via pin A2.

path that ends at the same output statement begins by sampling a previous value from pin C1 (line 79) into a software variable, `lastC1`, which is then compared with the current sample (line 92), thus also influencing whether or not the negative acknowlegement character is sent (line 95). Such alternative paths, due to conditional statements in the program code, were found to account for the many different high-to-low dataflow paths detected. Depending on the rigour of the security evaluation, the infosec evaluator may care to study each such path individually or may simply note the existence of potential data flow between the relevant microprocessor pins, regardless of its specific cause.

Finally, the acknowledgement character sent from the black microcontroller via its pin C1 reaches the black network via another AND gate (bottom left of Figure 12) and the black processor board's RS232 driver.

Paths such as this one, plus the various others produced in this case, again confirm the toolchain's ability to automatically identify complex dataflow paths which may be worthy of close scrutiny.

5.3 Some less obvious paths

Apart from the crucial paths between the data diode device's red and black serial ports, we also explored our toolchain's ability to identify other paths both within and through the device. In particular, we analysed the potential destinations of data emanating from the switches on the device's front panel, and possible sources of signals driving the front panel's LEDs. In practice the device itself would normally reside physically within a high-security domain, so there is no serious danger of an adversary receiving a coded message via the LEDs. However, the position of the switches could conceivably be detectable by an observer in the low-security domain, representing a more realistic threat.

For instance, SIFA's analysis, using our hardware model and the software model generated by the C-to-SIFA Converter, revealed 14 distinct dataflow paths from the 'ack mode' switch on the data diode device's front panel to the low-security serial port. As was the case for the acknowledgement bytes described above, this large number of paths proved to be due to the numerous conditional statements in the microcontrollers' programs that rely on the position of this switch. In essence, of course, the existence of these

```
68    output_high(PIN_C7);
69    getNextChar = FALSE;
70    #use rs232(baud=9600,Xmit=PIN_A1,
          Rcv=PIN_A0,parity=n,bits=8)
71    inputChar = getc();
73    output_low(PIN_C7);
```

Figure 14: Part of the black microprocessor's code responsible for reading data bytes (line 71) and flashing the 'ready to receive data' LED (lines 68 and 73).

paths simply confirms the obvious fact that an observer in the low-security domain can determine the position of the (high-security) 'ack' switch merely by noting the presence or absence of acknowledgements coming from the data diode device in response to bytes sent to it.

We also found numerous dataflow paths from the 'ack mode' switch to the two 'waiting for ack' LEDs on the data diode device's front panel (one LED is attached to each circuit board). This is to be expected because the position of this switch determines whether or not signals are sent to these LEDs. However, there were no paths from this switch to the 'ready to receive data' LEDs. Similarly, no paths were found leading from the red serial port to the 'ready to receive data' LED on the red circuit board since this port only receives acknowledgements, not data bytes. These results conformed precisely with our understanding of the data diode device's internal behaviour.

Some of the results were not so obvious, however. For instance, we were surprised to discover that the combined hardware-software analysis produced paths leading from the red serial port, which receives acknowledgement bytes only, to the 'ready to receive data' LED attached to the black circuit board, which displays the status of data bytes travelling in the opposite direction! Inspection of the black microcontroller's program code revealed that this interaction is due to the acknowledgement signals received by the black microcontroller from the red one controlling assignments to a software variable, `getNextChar`, which in turn is used to control code that sends signals to this LED (via black microcontroller pin C6). This was a good example of the C-to-Sifa Converter identifying paths not expected by the research team. (Furthermore, it was noted that the program code could be restructured to eliminate this flow, although in practice it does not represent a serious security threat given the assumption that the LEDs are accessible in the high-security domain only.)

A particularly counterintuitive finding was that no dataflow paths were produced from the black serial port to the black circuit board's 'ready to receive data' LED, which flashes once for each data byte received. The relevant part of the black microcontroller's program is shown in Figure 14. The LED is first switched on (line 68), the data byte is read (line 71), and the LED is then switched off (line 73). However, despite the clearly-evident *sequential* relationship between execution of these three statements, there is, in fact, no *dataflow* relationship between them. The byte read from pin A0 into variable `inputChar` is not sent to the LED connected to pin C7. Nor does the byte received control the signals sent to the LED; the same signals are sent to the LED regardless of the value of the byte received. Thus the C-to-Sifa Converter correctly produced no dataflow connection between inputs on pin A0 and outputs to pin C7 in this case. This is in accordance with the well-established principle of *noninterference* as a fundamental way of modelling information flow (Goguen & Meseguer 1982). There is, however, a *timing* relationship between these actions because the 'low' signal cannot be sent to pin C7 until the byte from pin A0 has been read. A 'timing channel' thus exists between these pins, but our toolchain does not (currently) attempt to perform timing analyses.

A similar case of an expected path *not* being found was from the red serial port, which receives acknowledgement bytes from the high-security domain, to the 'waiting for ack' LED attached to the red circuit board, which flashes to indicate to the operator that an acknowledgement is being processed. Again this finding by Sifa and the C-to-Sifa Converter was vindicated by manual inspection of the red microcontroller's program code, which showed that the same signals are always sent to flash this LED regardless of the value of the acknowledgement byte received. In fact, it is the position of the 'ack mode' switch that influences the behaviour of this LED, not the acknowledgement bytes themselves. Again there is a timing relationship between these actions, but no actual data flow.

6 Conclusion

One of the key steps during information security evaluations of communications devices is to trace all potential data flow through the device's circuitry and embedded program code. We have created a toolchain which automates this process by combining an existing circuitry analysis tool, Sifa, with a new analysis tool for embedded program code, the C-to-Sifa Converter. In this paper we have used a small, but complete, case study to show how this toolchain allows the flow of data to be traced seamlessly and accurately through a security device's hardware and software. At the time of writing we are conducting further case studies involving interrupt-driven microcontroller programs.

References

Ballance, R. A., Maccabe, A. B. & Ottenstein, K. J. (1990), The program dependence web: A representation supporting control-, data-, and demand-driven interpretation of imperative languages, *in* 'Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'90), New York, USA, June 20–22', ACM, pp. 257–271.

Cytron, R., Ferrante, J., Rosen, B. K., Wegman, M. N. & Zadeck, F. K. (1989), An efficient method of computing Static Single Assignment form, *in* 'Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'98), Austin, USA', ACM, New York, USA, pp. 25–35.

Fidge, C. J. & Corney, D. (2009), Integrating hardware and software information flow analyses, *in* 'Proceedings of the ACM SIGPLAN/SIGBED 2009 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2009), Dublin, June 19–20', ACM, pp. 157–166.

Goguen, J. & Meseguer, J. (1982), Security policies and security models, *in* 'IEEE Symposium on Security and Privacy', IEEE Computer Society, pp. 11–20.

ISO (2009), *ISO/IEC Standard 15408-1:2009, Information Technology—Security Techniques—Evaluation Criteria for IT Security—Part 1: Introduction and General Model*, 3.1 edn, International Organization for Standardization, Geneva, Switzerland.

Mahalingam, A., Butz, B. P. & Duarte, M. (2005), An intelligent circuit analysis module to analyze student queries in the Universal Virtual Laboratory, *in* W. Oakes, D. Voltmer & C. Yokomoto, eds, 'Proceedings of the 35th ASEE/IEEE Frontiers in Education Conference (FIE'05), Indianapolis, USA', Institute of Electrical and Electronics Engineers, New Jersey, USA, pp. F4E–1–F4E–6.

Mallen, S. (2003), Serial data diode device—Operation manual, Technical report, Defence Signals Directorate.

McComb, T. & Wildman, L. P. (2005), SIFA: A tool for evaluation of high-grade security devices, *in* C. Boyd & J. Nieto, eds, 'Proceedings of the Tenth Australasian Conference on Information Security and Privacy (ACISP 2005), Brisbane, Australia', Vol. 3574 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 230–241.

McComb, T. & Wildman, L. P. (2006), User guide for SIFA v.1.0, Technical report. Available from http://sifa.sourceforge.net/.

McComb, T. & Wildman, L. P. (2007), A combined approach for information flow analysis in fault tolerant hardware, *in* 'Proceedings of the Twelfth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2007)', IEEE Computer Society Press.

Sabelfeld, A. & Myers, A. C. (2003), 'Language-based information-flow security', *IEEE Journal on Selected Areas in Communications* **21**(1), 1–15.

Scholz, B., Zhang, C. & Cifuentes, C. (2008), User-input dependence analysis via graph reachability, *in* 'Proceedings of the Eighth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2008), Beijing, September 28–29', IEEE, pp. 25–34.