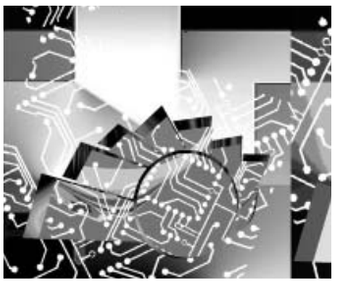


MÉTODO PARA LA EVALUACIÓN DE UN MICROCONTROLADOR DE NÚCLEO ABIERTO

METHOD FOR ASSESSMENT OF AN 8-BIT OPEN CORE MICROCONTROLLER

Sibilla B. Luz[1]
Christophe F. L. Bricout[2]
Ricardo P. Jasinski[3]
Volnei A. Pedroni[4]

Fecha de envió: Abril de 2011
Fecha de recepción: Mayo de 2011
Fecha de aceptación: Agosto de 2011

Resumen:

La etapa de verificación desempeña un papel fundamental en el diseño e implementación de microcontroladores. Con el fin de realizar una verificación acertada del diseño, son utilizadas algunas técnicas de verificación funcional tales como: pruebas definidas por el diseñador para verificar el desempeño ante casos extremos, la simulación a través de testbenches, y la ejecución de aplicaciones extensas. El proyecto propuesto en este trabajo tiene como objetivo desarrollar e implementar un método para la evaluación de un microcontrolador de núcleo abierto, con la realización de pruebas directamente sobre el hardware. Este enfoque presenta como ventajas, un proceso mucho más rápido que otros métodos que emplean simulaciones y menos requerimiento de memoria para las pruebas. Un Ethernet IP Core ha sido integrado al proyecto, con el fin de hacer que el método sea independiente del sistema operativo, de la arquitectura de microprocesador y de la herramienta de diseño.

Palabras clave:
Verificación, test bench, microcontrolador, Ethernet, IP core, VHDL.

Abstract:

The verification stage plays a major role in a microcontroller design and implementation project. An extensive series of tests must be performed looking for possible failures in the design. For this, some functional verification techniques are used, like manually devised tests targeting corner cases, simulation using sets of tests, and extensive sample applications. The project proposed in this paper aims to develop and implement a me-

1 MSc in Electrical Engineering. Federal University of Technology – UTFPR. E-mail: sibillabl@yahoo.com.br

2 PhD in Micro Systems. AXP Microeletrônica. E-mail: christophebricout@yahoo.fr

3 MSc in Electrical Engineering. Federal University of Technology – UTFPR. E-mail: jasinski@solvis.com.br

4 PhD in Electrical Engineering. Federal University of Technology – UTFPR. E-mail: pedroni@utfpr.edu.br

thod for the assessment of an open-source microcontroller, with the advantage of running tests directly on hardware. This approach makes the process much faster than other methods based only on simulations. Moreover, a smaller amount of memory is required for the tests. An Ethernet IP Core has been integrated to the project, in order to make the method independent from the operating system, microprocessor architecture, and design software. The test code is then sent through the network, enabling fast execution of a large number of test applications, whose output values can be compared against expected results or results obtained from simulation. The proposed method is supplementary to the conventional methods based on testbenches.

Key Words:
Verification, test bench, microcontroller, Ethernet, IP core, VHDL.

1. Introduction

Functional verification consists in testing all functionalities of a design, in order to guarantee that it operates according to the specifications. It is a very complex task, and more than half of the computer and human resources can be dedicated to this purpose in typical projects [1-3]. If the resulting implementation does not match the specified features, enormous commercial losses can easily occur [4].

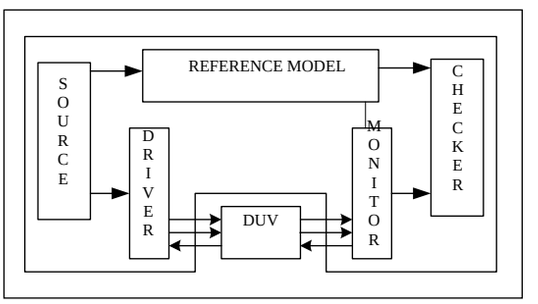
The verification process usually makes use of different techniques [5], including tests from different sources [6-7]. Initially, some tests are manually devised to target corner cases. Then simulations are run, including existing sets of tests. After the hardware is ready, extensive applications are executed. Finally, when an exhaustive test set in not practical or feasible, it is common to use pseudo-random stimuli. Conventional methods perform all the tests in the simulation environment, through testbenches. In this setup, the DUV (Device Under Verification) test results are automatically compared with a reference model provided by the user, as illustrated in figure 1 [8].

The proposed method has the advantage of performing these tests on hardware, making the verification time considerably shorter. However, the method can also be used as a complement to conventional methodologies, seeking better coverage in the microcontroller functional verification.

Several open-source projects, including microcontrollers, are available at the OpenCores website [9]. Many of them may be used in any application, while others present restrictions for commercial use.

However, since the available projects are not commercial versions (the latter undergo rigorous performance tests), correct functionality cannot be guaranteed. It is necessary to perform various additional tests to guarantee that the implementation has no errors. After a proper validation, the microcontroller can be used reliably in different projects.

Figure 1. Testbench to functional verification [8].



The goal of the proposed method is to facilitate the verification of 8-bit open-source microcontrollers. The confirmation of the success or failure of the test runs should be done through self-tests implemented by the user. However, verification techniques like pseudo-random stimuli can also be used. These tests must be provided by the user, and sent to a test circuit through the network.

2. Microcontroller Selection

The work described in this paper was part of a larger, industry sponsored project, which also included the selection of an adequate 8-bit microcontroller under the following requirements:

i) Availability of documentation, describing the implemented functions and design limitations.

ii) Availability of testbenches.

iii) The project should be an implementation of an existing, commercial product.

iv) Support tools for software development.

v) Preferably described in VHDL (as required by the sponsoring company).

vi) It should be a design with recent updates or corrections.

The first task was to do a search on the Internet, especially in the OpenCores website, to select the microcontroller to be used. Thirty-three microcontrollers were analyzed, trying to select the most complete 8-bit model, considering the following requirements: the project should be an implementation of an existing commercial product; with documentation describing implemented functions and design limitations; testbenches; software for data conversion to fill the program memory; project preferably described in VHDL; and, finally, a project with recent updates. Table 1 shows the surveyed designs, along with all relevant characteristics obtained from the project websites and source code. Looking at table 1, one sees that not all projects contained documentation, making it very difficult to know specific details.

Another important requirement is the availability of testbenches, which help understand the core operation, inspect simulation waves, and check signals and buses. Even though the majority of projects have testbenches, some are implemented in Assembly, not in a hardware description language. These software tests perform tasks such as printing of messages on a serial port, calculations, or reading a section of memory, and cannot be called proper testbenches.

In some cases, the MCUs had no equivalent commercial model. The JOP project, for example, implements a JVM (Java Virtual Machine) as a machine in hardware. Another example is the CPUgen project, which generates customizable RISC processors.

A data conversion utility is necessary to convert the implemented code in a format that can be used as the program memory. In some projects, such as the AVR_CORE, the converter generates, from a hex file, a VHDL file containing descriptions for a PROM (program memory) entity and architecture.

Among all surveyed candidates, four processor models were selected, with one or more implementations available for each of them. The selected microcontrollers included the Intel 8051, Zilog Z80, Motorola 6805, and Atmel AVR.

A more detailed examination of the documentation and code and a brief operation test were performed with the objective of selecting the final candidate. Eventually, the AVR_CORE was chosen [10]. This project is recent, described in VHDL, takes little FPGA space, has free C and Assembly compilers, and also a converter software (hex to vhdl) for the program memory. On the downside, it does not have testbenches.

3 AVR_CORE

After selecting the AVR Core project, several tests were performed on this microcontroller, using different codes written in C and in Assembly. Most AVR instructions were tested. Furthermore, some codes were implemented in different sequences of execution to determine whether any combination of instructions would cause any error.

Table 1. Evaluted microcontrollers.

Project	Arch.	Com. Model	Manufacturer	Doc.	Testbenches	Language	Converter[1]
68hc05	8-bit	mc68hc05	Motorola	No	No	VHDL	No
68hc08	8-bit	mc68hc08	Motorola	No	No	VHDL	No
AVR_CORE	8-bit	ATmega103	Atmel	Yes	No	VHDL	Yes
JOP	-	-	-	Yes	Yes	VHDL	Yes
RISCMCU	8-bit	90S1200	Atmel	Yes	Yes	VHDL	No
System09	8-bit	6809	Motorola	Yes	Yes	VHDL	Yes
System11	8-bit	68HC11	Motorola	No	Yes	VHDL	Yes
System68	8-bit	6800	Motorola	Yes	Yes	VHDL	Yes
T400	4-bit	COP420/421e COP410L/411L	National Semiconductor	Yes	Yes	VHDL	Yes
Marca	16-bit	-	-	Yes	No	VHDL	No
Risc5x	12-bit	PIC12 bit opcode	Microchip	Yes	Yes	VHDL	Yes
Mlite	8-bit	-	-	No	Yes	VHDL	Yes
CPUgen	-	-	-	Yes	Yes	VHDL	No
T51	8-bit	8052/8032	Intel	Yes	Yes	VHDL	Yes
T48 µController	8-bit	MCS-48	Intel	Yes	Yes		Yes
Tiny64	64-bit	-	-	No	Yes	VHDL	Yes
MiniMIPS	32-bit	MIPS I	-	Yes	Yes	VHDL	Yes
T80	8-bit	Z80 e 8080	Zilog/Intel	No	Yes	VHDL	Yes

1 Software to convert object file from C or Assembly compiler to VHDL file.

Figura 2 Formulario desarrollado para leer las variables (Visual Basic).

Project	Arch.	Com. Model	Manufacturer	Doc.	Testbenches	Language	Converter[1]
AX8	8-bit	90S1200 e 90S2313	Atmel	No	Yes	VHDL	Yes
Risc16f84	8-bit	PIC 16f84	Microchip	Yes	No	Verilog	Yes
PPX16 mcu	8-bit	PIC 16C55 e 16F84	Microchip	No	yes	Verilog	Yes
TV80	8-bit	Z80	Zilog	Yes	Yes	Verilog	No
AE18	8-bit	PIC18	Microchip	Yes	Yes	Verilog	No
C16	8-bit	8080	Intel	Yes	Yes	Verilog	Yes
Wishbone High P. Z80	8-bit	Z80	Zilog	Yes	Yes	Verilog	No
Aquarius	16-bit	CPU RISC SuperH-2 ISA	Motorola	Yes	Yes	Verilog	Yes
OpenRISC 1000	32-bit/ 64-bit	-	-	Yes	Yes	Verilog	Yes
8051 core	8-bit	8051	Intel	Yes	Yes	Verilog	Yes
MiniRisc	8-bit	PIC 16C57	Microchip	Yes	Yes	Verilog	Yes
YACC	32-bit	MIPS I	-	Yes	Yes	Verilog	Yes
S1 Core	64-bit	OpenSPARC T1 (Reduced version)	Sun Microsystems	Yes	Yes	Verilog	Yes
CPU8080	8-bit	8080	Intel	Yes	Yes	Verilog	Yes
aeMB	32-bit	Microblaze	Xilinx	Yes	Yes	Verilog	No

The model in figure 2 shows the sequence of operations performed for implementing the tests before the integration of the proposed method. First, the code written in C or Assembly is compiled. The object file generated by compilation is then used as input to a converter that provides a file containing the PROM memory entity and architecture in VHDL. The project must then be rebuilt and the program memory rewritten.

Figure 2. Original Model.

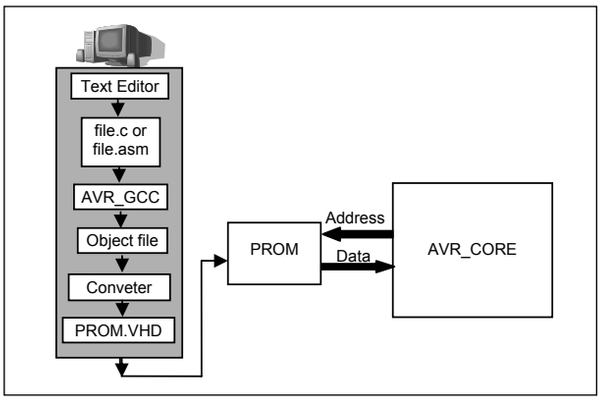
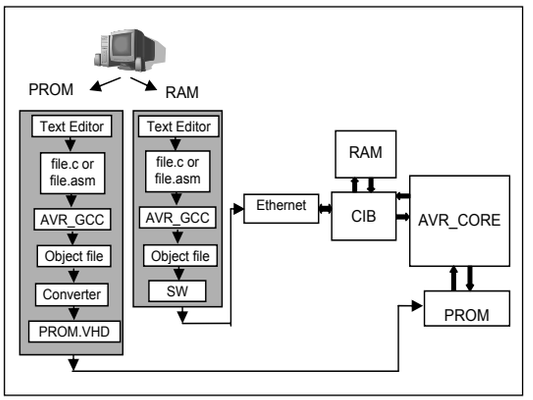


Figure 3. Proposed Model.



4. The Proposed Method

In the proposed method, the code to be executed by the microcontroller is sent in frames via the Ethernet network and stored in a RAM. After receiving the entire file, the microcontroller executes the memory contents, as shown in figure 3.

The PROM contents are not changed, and contain only data for the Ethernet startup configuration. The device with the microcontroller design is configured only once.

To minimize the design check time, and furthermore make the tests independent from the operating system, design software, and processor architecture, Ethernet communication is utilized. An Ethernet IP Core was used in the project, also provided by OpenCores, which has the Media Access Controller (MAC) of the Ethernet protocol implemented in Verilog [11].

The Ethernet network provides high speed data transfer and is available in most operating systems. Moreover, the user can choose any convenient tool for FPGA programming.

4.1. CIB - Control and Interface Block

The CIB block is responsible for the interface between the Ethernet IP Core and the AVR_CORE, and it contains the custom communication protocol implementation. Moreover, it performs the initial configuration of the Ethernet registers (operation speed, frame size limits, interrupts, frame discard, etc.).

When a frame containing the code for testing is received, the CIB records the data in a RAM. After completing this memory, the PROM is disabled and the microcontroller will execute the instructions from RAM.

The Ethernet registers are configured through the microcontroller ports A and B. Port A, shown in table 2, is used for control and contains the signals used for reading / writing registers. Port B is used for data.

Since the available ports are 8 bits wide and the Ethernet registers are 32 bits wide, the data transfer must be split into four parts. For this, a protocol was implemented, where the start signal indicates the beginning of the sequence. When this signal is high, it is known that the next four available values on port B form the

Table 2. Control Port.

Start	Program end	Status	-	-	-	Operation	Stop
7	6	5	4	3	2	1	0

address, while the next four parts are the data. The operation bit indicates registers writing or reading. The stop bit ends the operations sequence.

Bit 5 indicates the operation status, and can be changed by the test code to indicate success or failure.

To indicate the test code end, bit 6 is driven high, so instructions will be executed again from the PROM memory.

After establishing the connection between the computer and the test circuit, the microcontroller waits for the arrival of a new frame.

4.2. Memories Control

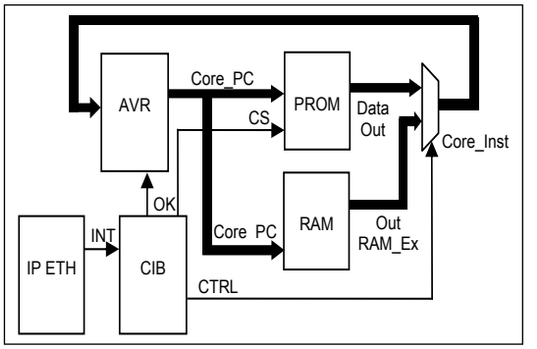
CIB also provides the multiplexing for the memories (PROM and RAM), using PC address and bit 6 from port A as parameters.

Addresses from 0000H to 01FF are used for RAM access. Addresses above 0200H are utilized for PROM access. If bit 6 from port A is high, that is, the test is over, microcontroller executes from PROM.

When a frame has been received, the Ethernet IP Core generates an interrupt, CIB writes FFH in port D, so the PC is set to 0008H. PROM's chip select is disabled, and from this point, instructions are being executed from RAM, as shown in the block diagram of figure 4. RAM addresses between 0000H and 0007H contain destination MAC address, source and size of the frame information.

Once the self-test sets bit 6 from port A to one, indicating that the test is finished, the PROM memory is able and the microcontroller return to execute from PROM, and wait for a new frame arrived.

Figure 4. Control Memories Block Diagram.

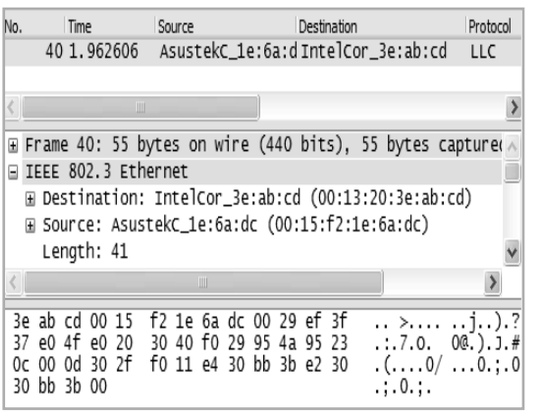


4.3. Data Transmission

The program Bittwist is used to send data to the Ethernet IP Core [12]. This program uses a packet capture (pcap) file obtained through WinDump, which is a network traffic analyzer able to capture and save to hard disk network packets [13].

To edit the captured packet contents, another program, named Bittwiste, is used [14]. With this editor, payload, source and destination addresses can be changed. The

Figure 5. Frame sent to test circuit.



desired layer and header can be specified too, among other options. In this project, the defined layer was the Ethernet layer, because the Ethernet IP Core does not have the TCP/IP protocol implemented.

Figure 5 shows an example of frame sent to the test circuit using a network analyzer.

5. Results

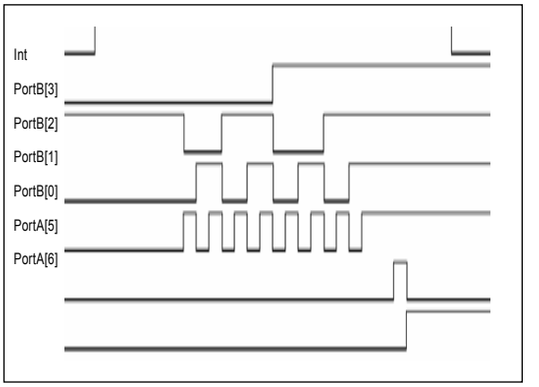
The first step after choosing the microcontroller was to find more about its operation. As the AVR_CORE had no testbenches, it was necessary to provide some simulation tests. The majority of the AVR instructions were verified using C and Assembly codes. All of the tests presented the expected results.

Before the Ethernet IP core integration, the CIB block was implemented and tested along with the AVR in simulations. After validating the CIB block, the Ethernet MAC was added to the project.

Because the original intent was to validate the AVR instruction set, more than 80% of the documented instructions were tested, using all of the available addressing modes. All of the arithmetic, logic, and branch instructions were exercised, and matched the expected results in all cases.

The development board, used for testing, was developed at the UTFPR Microelectronics Laboratory. This board has a Cyclone II FPGA EP2C8F256C8 and Ethernet PHY 78Q2123. One example of a self-test is presented in Figure 6, which shows signals captured with an oscilloscope to test the code presented in figure 7. In this test a register (r18) is cleared and then incremented 15 times. At the same time, the count value is

Figure 6. Test results captured with a logic analyzer.



replicated in at one of the microcontroller IO ports (port B). This self-test checks the register value and compares with the expected value, if the test is successful bit 5 of port A receives the value one. The last instruction sets bit 6 (port A) to one, returning the execution to the PROM. Bits 5 and 6 are continuously displayed by LEDs on the development board.

Figure 7. Assembly test (LST file).

```
.org $0008
000008    ef3f          ser TEMP
000009    bb3a          out DDRA, TEMP
00000a    bb37          out DDRB, TEMP
00000b    e04f          ldi r20, $0F
00000c    e020          ldi r18, $00
00000d    3040  loop:   cpi r20, $00
00000e    f029          breq check
00000f    bb28          out PORTB, r18
000010    954a          dec r20
000011    9523          inc r18
000012    940c          jmp loop
000013    000d
000014    302f  check:  cpi r18, $0F
000015    f011          breq ok
000016    e430          ldi r19, $40
000017    bb3b          out PORTA, r19
000018    e230  ok:     ldi r19,$20
000019    bb3b          out PORTA, r19
```

Table 3. Synthesis and simulation results.

	Original IP Cores (AVR + Ethernet)	Modified Cores + Test Control Hardware	Ratio
LUTs	4039	6577	162.8%
Registers and on-chip-memory bits	11010	18426	167.4%
Simulation time	2,262 s	0.1 s	0.0000442%

In order to evaluate the performance gain provided by the proposed method, a test run with a length of 5×10^6 clock cycles was executed in Mentor Graphics Modelsim simulator, which required 2,262 seconds (37.7 minutes). When executed in real-time in the FPGA, using the proposed method, this same test is run in only 100 ms (for a clock frequency of 50 MHz). These results indicate that, in this case, the proposed method is 22,620 times faster than a simulation of the same test code.

Table 3 summarizes the increase in logic resources usage (lookup tables – LUTs – and memory bits) with the adoption of the proposed method, as well as the corresponding speed gain in the aforementioned simulation run.

Finally, some faults were inserted in the microcontroller to demonstrate that the proposed method is capable of identifying errors in design. By examining test sequence outputs, the failing tests were identified and provided enough information to locate the fault in the original circuit.

6. Conclusions

The proposed method performs all necessary activities, without any additional delays in instruction availability when the program memories (PROM and RAM) are multiplexed. Consequently, errors due to system delays, which could affect the microcontroller verification, do not occur. All tests were performed at 50 MHz. The new test setup provided speed gains of up to 22,620 times compared to the simulation-only approach.

Additionally, the tests are independent from the design software, so the user can choose any tool for FPGA programming. The project is compiled and the FPGA is configured only once. Differently from tests using MIF (Memory Initialization File) files, for example, no specific tools are need here.

The tests are also independent from the operating system because most of them have Ethernet support. With the use of Ethernet to receive and transmit the test code through the network, the verification time is reduced considerably compared to traditional methods, which employ simulations.

The proposed method can also be used for other 8-bit microcontrollers, either RISC or CISC. It is only required that the microcontroller contains three 8-bit ports available. This method is supplementary and should be used in addition to methodologies based on testbenches in order to achieve better functional verification coverage.

References

[1] F. Casaubieith et. al., "Functional verification methodology of Chameleon Processor," Proceedings of 33rd Design Automation Conference, pp. 421-426, June, 1996.

[2] J. Kumar, C. Pixley, "Logic and functional verification in a commercial semiconductor environment," International Conference on Application of Concurrency to System Design, pp. 8-15, March, 1998.

[3] J. Bergeron, Functional verification of HDL models, 2nd ed. New York, NY: Kluwer Academic Publishers, 2002.

[4] L. Fournier, Y. Arbetman, M. Levinger, "Functional verification methodology for microprocessors using the Genesys test-program generator," Proceedings of Design, Automation and Test in Europe Conference, pp. 434-441, March, 1999.

[5] W. S. Encinas Jr, C. A. Dueñas, "Functional verification in 8-bit microcontrollers: A case study," Proc. of Symposium on Microelectronics and Devices, SBMicro 2001, pp. 1668-173, September., 2001.

[6] J. Monaco, D. Holloway, R. Raina, "Functional verification methodology for the PowerPC 604 microprocessor," Proc. of 33rd. Proceedings Design Automation Conference, pp. 319-324, June, 1996.

[7] M. Kantrowitz, L. M. Noack, "I'm done simulating; now what? Verification coverage analysis and correctness checking of the DECchip 21164 Alpha microprocessor", Proceedings of 33rd Design Automation Conference, pp. 325-330, 1996.

[8] K. R. G. da Silva, E. U. K. Melcher, G. C. S. Araújo, V. A. Pimenta, "An automatic testbench generation tool for a SystemC functional verification methodology," Proceedings of 17th Symposium on Integrated Circuits and Systems Design, SBCCI 2004, pp. 66-70, September, 2004.

[9] OpenCores, available at: http://opencores.org. Accessed: April, 2007.

[10] OpenCores, available at: http://opencores.org/?do=project&who=avr _core. Accessed: April, 2007.

[11] OpenCores, available at: http://opencores.org/?do=project&who=ethmac. Accessed: April, 2007.

[12] A. Y. C. Heng, "Bittwist," Faculty of Information Technology, Multimedia University, 2006. Available at: http://bittwist.sourceforge.net. Accessed: November, 2007.

[13] L. Degioanni, G. Varenni, F. Risso, B. John, "WinDump: tcpdump for Windows," 2006. Available at: ttp://www.winpcap.org/windump. Accessed: November, 2007.

[14] A. Y. C. Heng, "Bittwiste," Faculty of Information Technology, Multimedia University, 2006. Available at: ttp://bittwist.sourceforge.net. Accessed: November, 2007.