Project Number. GFP1205

Automated Testing For Mercury Computers

A Major Qualifying Project Report:

submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by:

Christopher Donnelly

Date: March 12, 2013

Approved:

Professor Gary F. Pollice, Major Advisor

1. Automated Testing
2. Linux
3. Component Discovery

Abstract

Mercury desires software that can self-discover the hardware components on the board and the system. Lshw+ is a tool developed from an open source program, lshw, capable of discovering components in a system and gathering all possible information into a database. Lshw+ reads certain system files to discover these components and gather all their information. It has a built-in capability to output this information into multiple formats, but more importantly, it saves all this information into an SQLite database.

Acknowledgements

I would like to thank Joe Plunkett and Jason Beauvais for allowing me to do this project and giving me access to systems to test lshw+.

I would also like to thank Professor Gary Pollice for helping me with the project and guiding me through any problems I encountered.

Table of Contents

List of Tables

1. Introduction

Mercury needs system level Acceptance Test (AT) software that provides 80-90% coverage at both a board and system level. The company desires software that can self-discover the hardware components on the board and the system. After the self-discovery, the software will choose the best test(s) to determine whether the component is operational and up to the customer's standards. The company has already developed many tests for these hardware components. Mercury tests and delivers systems on a mutually agreed acceptance test procedure. The mutually agreed ATP is designed to validate system functionality and performance characteristics that are important to the customer. The ATP is performed manually but is driven by an ATP procedure document that governs the testing process. If the procedure is followed correctly then it is by definition not error prone. The ATP document and tests developed for a particular system validates the requirements a customer has defined in their requirements to Mercury. The purpose of this project is to allow Mercury to identify testable units or functional blocks that they can then cross reference to their test database and identify areas that do not have test coverage. This project takes on the component discovery portion of the test software. The component discovery software will have to scan the system for all components and find as much information as possible on these components. This information will then need to be stored in a database for quick access later during the test selection process.

The remaining portions of this paper will include a background, methodology, results, and future work and conclusions. The Background section will go into more detail about Mercury. The Background also talks about automated testing and its importance to this project along with some history about automated testing. The Methodology explains the original requirements and how some of those requirements did not make it to the final product. It also includes the reasoning behind the database chosen and research for other known component discovery programs. The bulk of the Methodology section describes the work that was done to create the new component discovery software, how that software saves all the necessary information to a database, and the problems encountered and solved during the project. The Results section goes over what was completed for the project along with details into why certain aspects of the original requirements were removed from the project. Finally, the Future Work and Conclusions section explains the overall project results and how this author believes it can be expanded.

2. Background

2.1. Mercury

“Mercury is the leading supplier of very high-performance digital image, signal and sensor processing solutions for prime contractor customers in the defense industry, as well as the Intelligence Community” (Mercury Systems). Their expertise and capabilities across sensor processing enables their customers to minimize program risk and shorten time to deployment. Their success comes from the ability to provide open, commercially developed subsystems for the Intelligence, Surveillance and Reconnaissance (ISR) market. In addition to serving the defense industry, Mercury also creates applications in other areas such as homeland security, telecommunications and semiconductor manufacturing. Throughout Mercury’s 30-year history, they have worked with 26 prime contractors to execute more than 300 deployments on things such as Aegis, Global Hawk, JCREW, Patriot, Predator, and SEWIP.

Over the summer of 2012, the author worked as a co-op at Mercury, where he worked on three major projects. Working over the summer he learned about Mercury’s systems and their software. With that knowledge of their systems he could create a program capable of discovering both the common components in many computers along with the components that may only exist in Mercury systems.

2.2. Automated Test Coverage Analysis

As Mercury builds very large (with 10 to 100 elements) of systems with a very large number of permutations and combinations of hardware elements, it is difficult to ensure that a given set of hardware has complete coverage, or that hardware, when used in new combinations has tests that cover the particular configuration in question. Add to this the variables of firmware versions, hardware revisions, driver versions, and OS versions and there are a truly intimidating number of possible variables for any given system.

Complicating this problem is that the analysis for test coverage must actually be examined months or years before the system is actually built so that Mercury can assess the amount of work needed to augment a given test set to ensure adequate coverage for a given configuration. Therefore the initial need was to be able to evaluate the need for new tests to be created or for old tests to be modified (based upon software or firmware version changes). A substantial benefit to this type of analysis is that it can also lead to the ability to automatically select and execute tests.

All programmers make mistakes; the difference between the good ones and the bad ones is testing for these mistakes (Automated Tests). Testing for errors and mistakes in code is hard to do manually as the programmer has to check each and every test to make sure the expected output is the same as what the program outputs. Manual testing is also error prone and is not very effective at finding certain mistakes (Test Automation). This is where automated testing comes in; automated testing will run all of these tests and require almost no attention from the programmer. Automated tests will usually throw an exception or do something to tell the programmer there is a problem in a certain test. These tests can be run quickly and repeatedly every time the code is changed. Many times even minor patches to an application can cause other features to stop working (Test

Automation). Table 1 below shows many of the main benefits to automated testing (What are the Main Benefits of Test Automation?).

Test Automation Benefit	***Here's How***		
Save time : Generally increases the speed of the testing process and shortens the testing lifecycle	You can run scripts in the night, effectively giving you a day shift and night shift	You can run scripts on multiple machines-multiple platforms simultaneously	Scripts run faster than a manual tester
Increase quality: Through repeatability, reliability and comprehensiveness	Provides reusability and repeatability that can be run each time the application is changed	Accuracy in testing is increased by reducing possibility of human error and making less dependent on individual capabilities	Facilitates creation of tests that check all aspects of the application leading to an overall increase in quality
Utilize manpower more effectively: Apply your skills and time where they are needed most	Manual testing can be used for new feature validation while automation can be used for regression	Most functional and regression testing can execute unattended	Focus efforts where you didn't have time for before, such as performance and security
Increase coverage: You can test how the software reacts under repeated execution of the same operations	Do the same tests on multiple Configurations, more configurations, more benefit here	Regression suites can cover every feature in your application which may be difficult to accomplish manually depending on your application's size	Different versions can reuse the same automation code with minor modifications
Programmable*:* You can program sophisticated logic	Some web applications use hidden information like session id, and account information which are impossible or difficult to manually verify	Some applications have no UI, and are best suited or can only be tested with automation rather than manual testing	You can program logic to respond differently to different environments and situations

Table 1. Benefits of Test Automation

3. Methodology

3.1. First Requirements

The original objective was to write a program that would discover all the components on a motherboard and store as much information about them as possible in a database. These boards are embedded systems so the program could not involve large libraries due to memory limitations. The database this program would need to create would also have to be fairly small and easy to use. After storing the information, a separate program would use this database along with a test database to select tests for the board. These tests would help Mercury validate their boards are functioning correctly.

Along with selecting tests the original specifications included a website that would print all the information from the component database. This was too much to be completed in a single project; the scope was revised and the component discovery portion was chosen to be the only deliverable.

3.2. Initial Thoughts and Plans

One solution considered using built-in Linux commands to discover these components and record all the information. That would involve building a parser to pull all the important information out of the output of these commands. As for the database, the two main choices suggested were MySQL and SQLite. MySQL has several advantages over SQLite. MySQL is a database server with different ways of storing its data and the ability to serve its data out to incoming requests. It also has many additional features like: user permissions, performance tuning, and the ability to scale in large applications, however, it is large. SQLite is a database based on a single file and is useful in embedded systems. It does not have any of the MySQL features mentioned above (SQLite vs MySQL). SQLite is easy to setup and only needs one library included in the program to work. Since this project is being built for embedded systems and user permissions and scaling are not issues, SQLite was clearly the better choice.

3.3. Research

Instead of initially starting out writing the program that uses the built-in Linux commands, research was done on other options. Research showed that multiple programs exist, but one of them stood out above all others. Lshw (HardWareLiSter) is an open source program that is capable of discovering the components of most Linux systems (HardwareLiSter (lshw)). Lshw is easy to implement on the embedded systems and does not require any libraries other than default libraries already installed on the system. This program was selected to complete this project.

3.4. Creating Lshw+

After first tests at Mercury, lshw showed all components that a normal Linux system would include. Unfortunately, Mercury had some of their own components on this board and they were not found. This meant lshw needed to be expanded to include these components and still needed a way to store all the information.

Lshw has three main outputs for its information: HTML, XML, and plain text. At first thought, the XML output could be used along with a parser to grab all the information and then could be stored into a database. Upon further investigation there turned out to be a dump file in the source code for lshw. Inside this dump file was mentions of an SQLite database and pre-written commands to create tables and store data. Unfortunately, not all of these commands were written and there was no mention of this dump option in the help menu when running lshw. This dump file is only included in the program if the Makefile is told to build lshw with SQLite. The Makefile also tells the main file whether or not to include the dump option in the help menu. Fixing the Makefile introduced many errors that had to be fixed before continuing. The main file only needed to activate the dump option so most of the errors were coming from the dump file itself. Most of these errors from the dump file were due to the fact that the code was not completely written. Once all this code was completed, lshw built with no errors and the dump option worked perfectly. The database portion of the project was complete and now all that was left was the final Mercury components and information towards those.

HTML, XML, and plain text outputs would only show certain components in the system and hide some others that lshw had found. In the database was a mention of four unknown components and these components had a Mercury vendor label. Apparently, Mercury does not give Linux much information as far as a device or class id for these components so they all came up as unknown devices and were not included in any other output. For this particular board, the only Mercury components missing were two Serial RapidIO (SRIO) boards that actually allow this board to communicate to other boards in the entire system. The SRIO board appears multiple times because there are two different field-programmable gate arrays (FPGAs) for each board, the “IOM” and the “POET”. The only way to tell the difference between the two was a sub_device id. This id could be found using Linux commands outputting the highest amount of detail. Since these commands read the same system files that lshw does; lshw should have all the information somewhere in the database.

Lshw creates multiple tables in its database. One of them being the components and most of the information needed; the rest being extra information for each component that is not always needed or useful. One of these tables gave sub_device ids which were all that was needed to determine the “IOM” and “POET” FPGAs. Lshw had to be changed to include this data in the main component table. This information was then used to update the device descriptions of the Mercury components. Each description was originally “Unknown class” but was changed respectively to “IOM FPGA” and “POET FPGA.” With that done, lshw+ was finished and all that was needed was more testing. Once testing was complete, lshw+ performed wonderfully and discovered all components for all the boards tested.

4. Results and Analysis

4.1. Lshw+

Lshw+ worked wonderfully; after the second round of testing Mercury boards, it proved successful and discovered all the components on the board. All the information was stored in an SQLite database for future use. Lshw+ works well with embedded systems as it is small and does not rely on large external libraries. This project was a success and will lead Mercury closer to having their finished testing software.

4.2. Remains of the testing software

At the beginning of the project, the requirements were revised so that this project could be finished within the time constraints. The test selection portion of the original requirements was removed because Mercury did not have a database of their tests to compare with the component database. The website was removed to be used in a separate Mercury project. Lshw+ and the component database are being used in this website to give the user a graphical look at the components on the board.

The project may not have been completed as originally planned; it is still a big step towards the final product for Mercury. The component discovery was completed successfully including all of the Mercury-specific components. Finishing up the final portions of the project will involve work on the test database and the test selection software.

5. Future Work and Conclusions

There is much of work that can come out of this project; lshw+ can be expanded even more to include other systems and possibly expand to support other output formats or database formats instead of just SQLite. Mercury will most likely continue on this project to finish up their original plan of a testing system that will discover all components and select tests suitable for each board. Future work would have to setup the test database along with creating the test selection program. Many of the tests will be easy to add to the database. Some tests have certain criteria to be selected for components. These specific criteria will make creating this database a challenge. Mercury could also expand this project farther and have the system automatically run all the tests to provide more confidence their products are working correctly and speed up the entire process of testing their products. Mercury currently runs their ATP software manually; if the procedure document is followed correctly then there are no errors in testing but if something is not done correctly then some aspect of this document can be skipped. Someone in the future could automate this procedure to guarantee no possible errors in the ATP testing. The website was already expanded to not only create a graphical view of the components database but also a system manager to allow for quick views into temperature readings and hard drive diagnostics.

This project was successful even though not everything originally planned was completed. The component discovery portion of this project was finished completely which gave Mercury a large step towards being done with their automated testing system.

The best part of lshw+ is that it can be used in future versions of Linux because system files that contain this information will not change.

Appendix A: LSHW+ Documentation

DOCUMENTATION FOR LSHW+

AUTHOR: CHRISTOPHER DONNELLY

VERSION 1.0

Building the Program

LSHW+ is simple to build. Just move to the lshw+ directory and run make there or in the src directory inside. The lshw+ executable will appear in the src directory and can be moved anywhere to run. This executable may also be built on one system and copied to others without any problems or need to rebuild.

```
cd lshw+
make
```

or

```
cd lshw+
cd src
make
```

Running LSHW+

LSHW+ does not have many option so it is pretty easy to run. Only the basic output formats along with the database dump will be discussed in this documentation but if there are any other options you would like to explore, you can read LSHW+ help page.

```
lshw –h
```

The simplest way to run this program is to just execute the command with no options but there is not much usable information there. LSHW+ outputs the information in text format onto the terminal screen.

```
lshw
```

XML Output

```
lshw -xml >  <file_name>
```

HTML Output

```
lshw -html >  <file_name>
```

SQL Dump

```
lshw -dump <sql db name>
```

XML and HTML outputs will output their information straight to the terminal window so it needs to be piped to a file. The dump option will take a database name and create a database in the working directory of all the information stored. This database contains 5 tables: nodes, capabilities, configurations, hints, and logicalnames. Lshw+ has been updated for a website view in use with WT webtoolkit so there are three columns in the nodes table that contain information from configurations, hints and logicalnames in order to keep the web view simple and easy to work with. This setup makes these three columns fairly unreadable.

The Database

There is one main file in the source that deals with the database: dump.cpp. This file deals with all the database dumping commands and can be modified to match certain database needs. All commands are pre-written and just require binding certain variables.

The Dump File

The first function in the dump file handles creating the tables and putting information into the “META” table. The META table includes basic information like the application name and the name of the operating system.

```
static bool createtables(database & db){
try {
db.execute("CREATE TABLE IF NOT EXISTS META(key TEXT PRIMARY KEY COLLATE
NOCASE, value BLOB)");
```

Create the META table.

```
statement stm(db, "INSERT OR IGNORE INTO META (key,value) VALUES(?,?)");
```

Setup the command that will insert data into the META table.

```
stm.bind(1, "schema");
stm.bind(2, 1.0);
stm.execute();


stm.reset();
stm.bind(1, "application");
stm.bind(2, "lshw+");
stm.execute();


stm.reset();
stm.bind(1, "creator");
stm.bind(2, "lshw+/" + string(getpackageversion()));
stm.execute();


stm.reset();
stm.bind(1, "OS");
stm.bind(2, operating_system());
stm.execute();


stm.reset();
stm.bind(1, "platform");
stm.bind(2, platform());
stm.execute();
```

These lines bind variables to the two different positions in the above command. Reset will clear the previous binding so execute does not add incorrect data.

This portion handles creating all the tables. The most important part is the first table. This table is the table of all the nodes and their information.

```
db.execute("CREATE TABLE IF NOT EXISTS nodes(path TEXT PRIMARY KEY, id TEXT NOT NULL COLLATE NOCASE, parent TEXT COLLATE NOCASE, class TEXT NOT NULL COLLATE NOCASE, enabled BOOL, claimed BOOL, description TEXT, vendor TEXT, product TEXT, version TEXT, serial TEXT, businfo TEXT, physid TEXT, slot TEXT, size INTEGER, capacity INTEGER, clock INTEGER, width INTEGER, dev TEXT, logicalname TEXT, configurations TEXT, hints TEXT)");

db.execute("CREATE TABLE IF NOT EXISTS logicalnames(path TEXT PRIMARY KEY, logicalname TEXT NOT NULL, node TEXT NOT NULL COLLATE NOCASE)");

db.execute("CREATE TABLE IF NOT EXISTS capabilities(capability TEXT NOT NULL COLLATE NOCASE, node TEXT NOT NULL COLLATE NOCASE, description TEXT, UNIQUE (capability,node))");

db.execute("CREATE TABLE IF NOT EXISTS configuration(config TEXT NOT NULL COLLATE NOCASE, node TEXT NOT NULL COLLATE NOCASE, value TEXT, UNIQUE (config,node))");

db.execute("CREATE TABLE IF NOT EXISTS hints(hint TEXT NOT NULL COLLATE NOCASE, node TEXT NOT NULL COLLATE NOCASE, value TEXT, UNIQUE (hint,node))");

db.execute("CREATE TABLE IF NOT EXISTS resources(node TEXT NOT NULL COLLATE NOCASE, type TEXT NOT NULL COLLATE NOCASE, resource TEXT NOT NULL, UNIQUE(node,type,resource))");

db.execute("CREATE VIEW IF NOT EXISTS unclaimed AS SELECT * FROM nodes WHERE NOT claimed");

db.execute("CREATE VIEW IF NOT EXISTS disabled AS SELECT * FROM nodes WHERE NOT enabled");
}
catch(exception & e)
{
return false;
}
return true;
}
```

The next function is the actual dump function that handles putting the rest of the information into all the tables we just created. Lshw+ calls each component a node.

```
bool dump(hwNode & n, database & db, const string & path, bool recurse){
if(!createtables(db))
return false;
```

Try to create the tables. If it fails then return.

```
try {
unsigned i = 0;
statement stm(db, "INSERT OR REPLACE INTO nodes (id,class,product,vendor,description,size,capacity,width,version,serial,enabled,claimed,slot,clock,businfo,physid,path,parent,dev) VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
string mypath = path+(path=="/"?"":"/")+n.getPhysId();
```

Setup the command to add the first set of information into the nodes table.

```
stm.bind(1, n.getId());
stm.bind(2, n.getClassName());
if(n.getProduct() != "") stm.bind(3, n.getProduct());
if(n.getVendor() != "") stm.bind(4, n.getVendor());
if(n.getDescription() != "") stm.bind(5, n.getDescription());
if(n.getSize()) stm.bind(6, (long long int)n.getSize());
if(n.getCapacity()) stm.bind(7, (long long int)n.getCapacity());
if(n.getWidth()) stm.bind(8, (long long int)n.getWidth());
if(n.getVersion() != "") stm.bind(9, n.getVersion());
if(n.getSerial() != "") stm.bind(10, n.getSerial());
stm.bind(11, (long long int)n.enabled());
stm.bind(12, (long long int)n.claimed());
if(n.getSlot() != "") stm.bind(13, n.getSlot());
if(n.getClock()) stm.bind(14, (long long int)n.getClock());
if(n.getBusInfo() != "") stm.bind(15, n.getBusInfo());
if(n.getPhysId() != "") stm.bind(16, n.getPhysId());
stm.bind(17, mypath);
if(path != "") stm.bind(18, path);
if(n.getDev() != "") stm.bind(19, n.getDev());
stm.execute();
```

Bind the variables for the information about this component and execute the command to add the information to the table.

```
stm.prepare("INSERT OR REPLACE INTO logicalnames (node,logicalname) VALUES(?,?)");
vector<string> keys = n.getLogicalNames();
for(i=0; i<keys.size(); i++)
{
stm.reset();
stm.bind(1, mypath);
stm.bind(2, keys[i]);
stm.execute();
}
//Take what was just put into logicalnames and put it into nodes.
//Slight difference. This is setup for internet view. <br /> for the line break.
stm.prepare("UPDATE nodes SET logicalname=? WHERE path=?");
stm.reset();
stm.bind(2, mypath);
string lnames = "";
for(i=0; i<keys.size(); i++)
{
if( i == 0 ) lnames = keys[i];
else lnames += "<br/>" + keys[i];
}
stm.bind(1, lnames);
stm.execute();
```

Add logicalname information to the logicalnames table and the logicalnames column of the nodes table.

```
stm.prepare("INSERT OR REPLACE INTO capabilities (capability,node,description)
VALUES(?,?,?)");
keys = n.getCapabilitiesList();
for(i=0; i<keys.size(); i++)
{
stm.reset();
stm.bind(1, keys[i]);
stm.bind(2, mypath);
stm.bind(3, n.getCapabilityDescription(keys[i]));
stm.execute();
}
```

Add capabilities information to the capabilities table.

```
stm.prepare("INSERT OR REPLACE INTO configuration (config,node,value) VALUES(?,?,?)");
keys = n.getConfigKeys();
for(i=0; i<keys.size(); i++)
{
stm.reset();
stm.bind(1, keys[i]);
stm.bind(2, mypath);
stm.bind(3, n.getConfig(keys[i]));
stm.execute();
}


//Place configurations into the nodes again with html web view.
stm.prepare("UPDATE nodes SET configurations=? where path=?");
stm.reset();
stm.bind(2, mypath);
string myconfigs;
for(i=0; i<keys.size(); i++)
{
if(i==0) myconfigs = keys[i] + "=" + n.getConfig(keys[i]);
else myconfigs += "<br/>" + keys[i] + "=" + n.getConfig(keys[i]);
}
stm.bind(1, myconfigs);
stm.execute();
```

Add configuration information to both the configuration table and the configuration column of the nodes table.

```
stm.prepare("INSERT OR IGNORE INTO resources (type,node,resource) VALUES(?,?,?)");
keys = n.getResources(":");
for(i=0; i<keys.size(); i++)
{
string type = keys[i].substr(0, keys[i].find_first_of(':'));
string resource = keys[i].substr(keys[i].find_first_of(':')+1);
stm.reset();
stm.bind(1, type);
stm.bind(2, mypath);
stm.bind(3, resource);
stm.execute();
}
```

Add resource information to the resource table.

```
stm.prepare("INSERT OR REPLACE INTO hints (hint,node,value) VALUES(?,?,?)");
keys = n.getHints();
for(i=0; i<keys.size(); i++)
{
stm.reset();
stm.bind(1, keys[i]);
stm.bind(2, mypath);
stm.bind(3, n.getHint(keys[i]).asString());
stm.execute();
}



//Place hints into the nodes database.
stm.prepare("UPDATE nodes SET hints=? where path=?");
stm.reset();
stm.bind(2, mypath);
string myhints;
for(i=0; i< keys.size(); i++)
{
if(i==0) myhints = keys[i] + "=" + n.getHint(keys[i]).asString();
else myhints += "<br/>" + keys[i] + "=" + n.getHint(keys[i]).asString();
}
stm.bind(1, myhints);
stm.execute();
```

Add the hints information to the hints table and the hints column in the nodes tables.

Hints are extra information for each node like the device and sub_device id.

```
stm.prepare("INSERT OR REPLACE INTO hints (hint,node,value) VALUES(?,?,?)");
stm.reset();
stm.bind(1,"run.root");
stm.bind(2,"");
stm.bind(3,(long long int)(geteuid() == 0));
stm.execute();
stm.reset();
stm.bind(1,"run.time");
stm.bind(2,"");
stm.bind(3,(long long int)time(NULL));
stm.execute();
stm.reset();
stm.bind(1,"run.language");
stm.bind(2,"");
stm.bind(3,getenv("LANG"));
stm.execute();
```

Add extra information to the hints table; whether or not lshw+ was run as root and the time.

This final portion of code will recall the dump function to run on this node’s children and continue to run through all the nodes in the system.

```
if(recurse)
for(i=0; i<n.countChildren(); i++)
dump(*(n.getChild(i)), db, mypath, recurse);
}
catch(exception & e){
return false;
}
return true;
}
```

The Nodes Table

The Node Table is explained in the original lshw website (http://ezix.org/project/wiki/HardwareLiSter). The information follows:

lshw displays nodes with attributes in a tree-like structure. Each node has its individual status: it be*CLAIMED* (potentially usable) or *UNCLAIMED* (no driver has been detected for this node), *ENABLED* (this device is supported and can be used) or *DISABLED* (this device is supported but has been disabled).

Attribute	**Meaning**	**Example 1**	**Example 2**	**Example 3**
id	internal identifier used by lshw	cpu:2	network:1	cdrom:0
class	device's class (see	processor	network	disk

	below)			
description	human-readable description of the hardware node	CPU	Ethernet interface	DVD reader
vendor	vendor/manufacturer of the device	Intel Corp.	Advanced Micro Devices [AMD]	
product	product name of the device	Intel(R) Pentium(R) 4 CPU 1.90GHz	79c970 [PCnet32 LANCE]	Hewlett-Packard DVD Writer 100
version	version/release of the device	15.1.2	25	1.37
serial	serial number of the device		00:60:b0:87:86:22	CN1AA0786J
capacity	maximum *capacity* reported by the device		100000000 (100MB/s)	
size	actual *size* of the device	1900000000 (1.9GHz)	10000000 (10MB/s)	
clock	bus clock (in Hz) of the device	100000000 (100MHz)	33000000 (33MHz)	
width	address width of the	32	32	

	device (32 or 64 bits) this has nothing to do with having 32bit or 64bit driver			
slot	slot where the device is connected	Processor 1		
logicalname	logical name under which the node is known to the system		eth0	/dev/hdc
dev	device number (*major.minor*)			22d:0d
businfo	bus information	cpu@0	pci@02:0a.0	ide@1.0
physid	physical id	4	a	0

References

Automated Tests. (n.d.). Retrieved February 2, 2013, from PHPUnit: http://www.phpunit.de/manual/current/en/automating-tests.html

HardwareLiSter (lshw). (n.d.). Retrieved February 22, 2013, from ezix: http://ezix.org/project/wiki/HardwareLiSter

Mercury Systems. (n.d.). *Who We Are*. Retrieved February 22, 2013, from Mercury Systems: http://www.mrcy.com/company-information/who-we-are/

SQLite vs MySQL. (n.d.). Retrieved February 22, 2013, from All About Programming: http://schimpf.es/sqlite-vs-mysql/

Test Automation. (n.d.). Retrieved February 22, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Test_automation

What are the Main Benefits of Test Automation? (n.d.). Retrieved February 22, 2013, from XBOSOFT: http://www.xbosoft.com/test-automation-benefits/