To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

H8/300L SLP Series

Debugging Technique (DebugTec)

Introduction

This document will elaborate the various types of bugs that developers made.

Next, the different classification of testing to detect bugs will be highlighted.

Finally, the various effective ways of error prevention, error detection and debugging techniques will be discussed.

Target Device

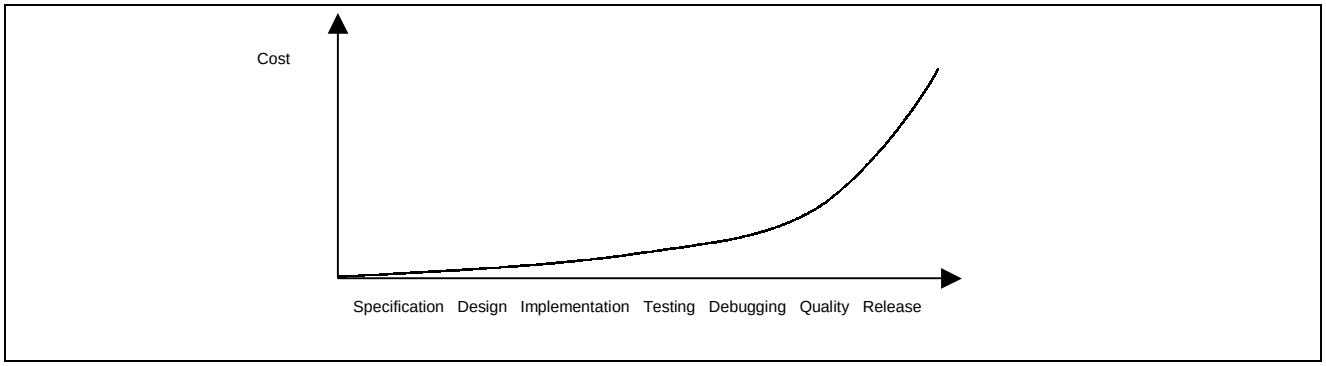
All H8/300L SLP Series MCU

Contents

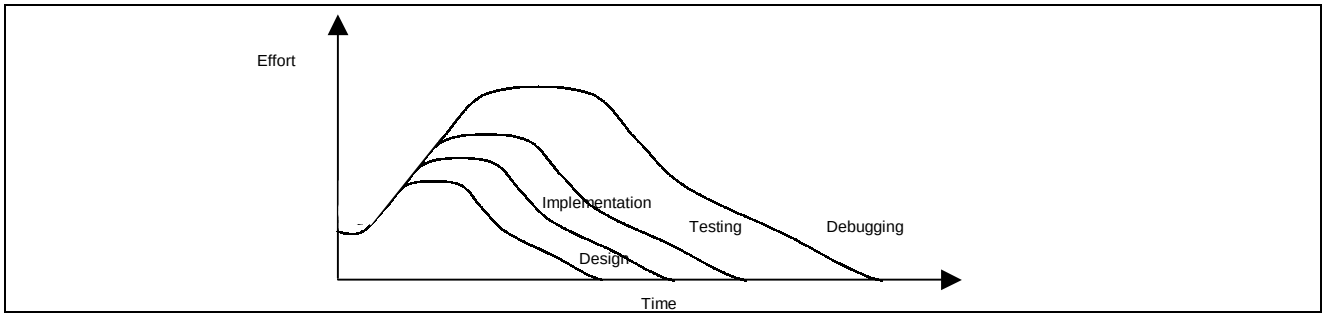
1. Introduction

Bug fixing is an inevitable activity during a development cycle. It is not an activity that is carried out solely at the end of the development. Developers have to pay continuing effort in bugs prevention, detection, and correction.
Two undeniable statistics will help developers to understand what strategy to uses, in order to have an efficient and effective bug fixing process.

1. The cost of fixing bugs is higher at the end of the development cycle.



2. More effort is required during integration (testing and debugging) than implementation stage.



The implementation stage always takes shorter time to complete than the integration stage. The integration stage took up longer hours because this is the stage whereby most bugs are discovered. Statistic showed that with a clean compile, there is one bug detected per 20 lines of code written. i.e., a 32-k code will produce an approximate 1500 errors of code.

In this document, various topics on error prevention, which happen during the design and coding stages, will be discussed. The testing methodologies to detect bugs are also highlighted.

The objective is to define and clearly state the various stages of work during development. This will allow developers to have a systematic plan to prevent, detect, and correct their bugs.

Testing is the systematic exercise of a program, which is believed to be correct. An effort to ensure bug does not exist.

Debugging is the process of identification of a program fault, whereby the error existence is known.

Most topics covered are software-related. However there is always a stage whereby hardware is one of the suspicious culprits. In the later section, various techniques on hardware troubleshooting will also be covered.

2. Bug Types

In order to identify and remove bugs, bugs must be understood.

The following are the general classifications of bugs that programmers may encounter:

1. Non-implementation error sources
   - Ambiguous/unclear specification
   - No handling of exception
2. Algorithm/logic/ processing bugs
   - Condition loop that execute one more extra loop
   - Logical error in AND and OR condition
   - Misunderstood the complex algorithm.
3. Data bugs
   - Pointer error
   - Data range overflow/underflow
   - LSB and MSB definition
   - Semantic
4. Real-time bugs
   - Interrupt handling and suppression
   - Task synchronization
5. System bugs
   - Stack overflow/underflow
   - Resource sharing problem
6. Other bugs
   - Syntax/typo
   - Memory leak
   - Peripheral initialization

3. Design & Coding Stage - Bug Prevention & Debugging Facility

Good coding practices and plans at initial stage will reduce bug-fixing process.

There are no hard and fast rules, but the following will highlight various topics on bug prevention, these will be further elaborate in the later sections.

1. Follow a good coding standard
2. Program defensively
3. Plan for error handling
4. Plan for debugging and testing
5. Define variable correctly (unsigned/signed integer long)
6. Place all components in a known initialized state when startup.
7. Place meaningful and correct remarks in software codes.
8. Plan for the use of debugging tool.
9. Plan for monitor/debugging code.
10. Use of Printf and Assert.
11. Prepare test pins and pads for testing - clock, ground Vcc, IRQ, external trigger, and others.
12. Mark (silk screen) pin numbers, signals and jumper names.

4. Testing Stage - Bug Detection

There are three different classification of testing:

1. First stage
   — Testing for the initial coded program to reach a functional stage
   e.g. unit test
2. Quality Assurance stage
   — Testing done on the completed system to reach a “fool-proof” stage
   e.g. coverage test
3. Maintenance or regression testing
   — Testing aim to ensure that no new bugs are introduced into the tested system during the extension and repair
   e.g. self test or custom made test done during the Quality Assurance stage

All tests done in the first and the quality assurance stages shall be properly stored and documented as these formed the basis for the maintenance or regression testing stage.

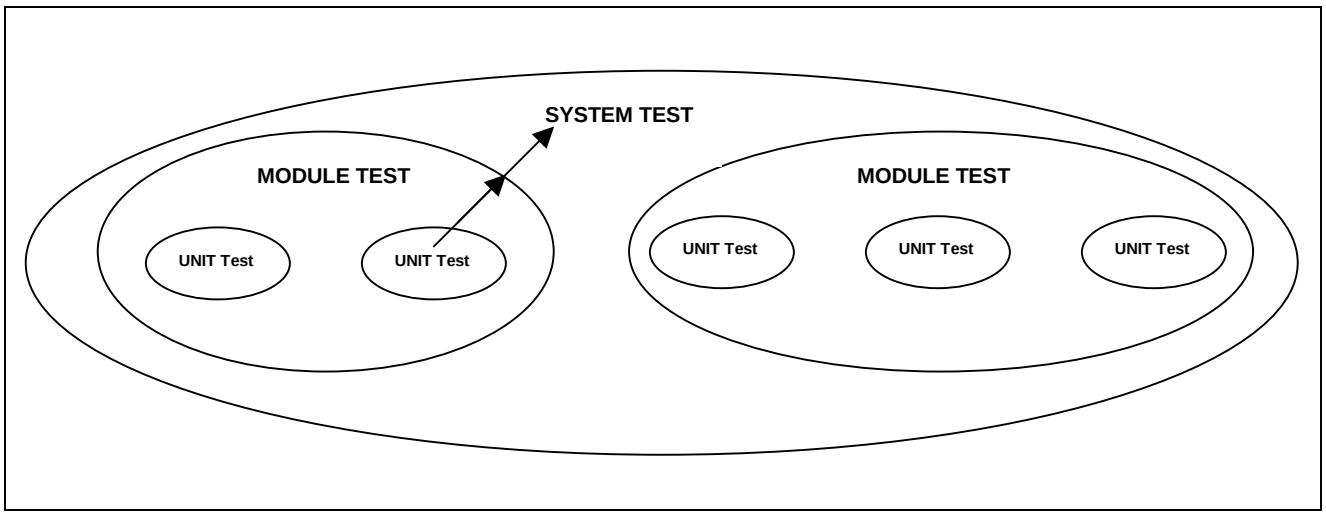
These tests can be carried out in two manners:

1. Statically
2. Dynamically

Before addressing the detail of tests, the object to be tested must be clearly defined.

1. Unit test
2. Module test
3. System test

It is always more difficult to debug a complete system than a smaller module. This is due to the many possible combinations of errors. Thus it is always important to begin the test with a smaller unit, before proceeding to a bigger module. This will lead to a shorter and easier debugging stage, at the final integration system test



4.1 Static Test

A static test is the simplest test. There is no requirement to execute the code. However, the basic checking process may lead to the detection of many bugs.

1. Code walk through

Developers shall be able to analyze their code while reading through the code that was written.

- Sequence of flow
- Paths taken
- Passing arguments

e.g. May discover logical errors such as omission of brackets.

2. Compiling the code

Code compilation will enable the check on the correctness of the syntax. It is important not to ignore the asserted warning by the compiler.

e.g. Constant Section not used may signify that wrong declaration of variables.

3. Analysis Tool – HEW Call Walker and Map Viewer

There are tools that help developer in their analysis. The Renesas High-performance Embedded Workshop (HEW) provides these tools, such as Map Viewer and Call Walker [Note], for the developers to check on the compiled code.

e.g. Map Viewer may show that code is placed in the wrong section

4. Evaluation with design specification

It is a good practice to check the coding with reference to the initial specification laid. This will ensure a better integration in the later stage.

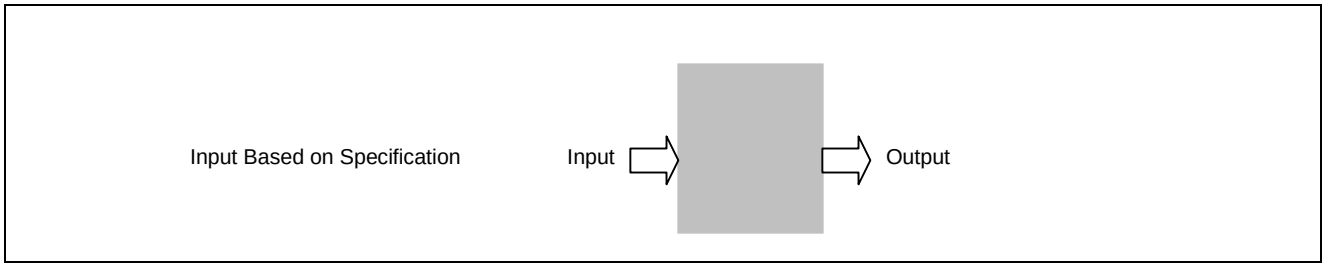
e.g. A wrong assumption of hardware register definition may cause a system hang without any symptom.

Note: Map Viewer: Provide a graphical visualization of where the compiled code will be located (HEW built-in component)
Call Walker: Show the depth of stack required by the analyzing the path of functions called.

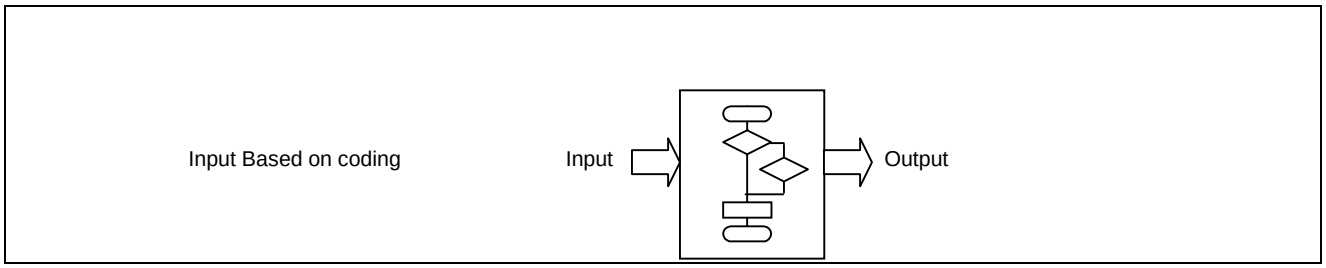
4.2 Dynamic Test

Code written for each unit, module and system will be executed to evaluate its working condition. The tests can be classified into

1. Black box test (functional testing)
   A. Compare test program behavior against a requirement specification
   B. Examine the program accomplishment without regard to its working methodology



2. White box test (structural testing)
   A. Compare test program behavior against the apparent intention of the source code
   B. Examines how the program works, taking into account possible pitfalls in the structure and logic



Tests can be carried out in various means:

1. Software execution
2. Test-script generation
3. Software performance
4. Data correctness verification
5. Emulation of target system

Generally a debugger tool such as the emulator or simulator must be used in the testing. Codes are tested by single stepping through the program, or via **execution** from one point to another. The input can be easily manipulated in a debugger environment such as HEW registers/ memory/ IO/ local variable/ windows. The output can be monitored via the same window or through the Trace window, whereby the detailed coverage is shown. This can be automated through the HEW **test script**; TCL/TK. This test can be stored and reused to re-evaluate the integrity of the code. Moreover the HEW debugger coverage and performance analysis will enable developer to have a more efficient **software performance** evaluation. Facility such as memory compare and file verify are good tools for verifying **data correctness**. For the target system behavior test, the usage of the **emulator** is inevitable.

After the completion of tests, it is recommended to have a beta-site tester (or a third person) to use the system before the actual system is launched. This is to ensure a wider coverage of tests from another perspective.

4.3 Coverage

An important testing concept is coverage.

There are various terms used, such as:

Condition testing: execute all logical conditions

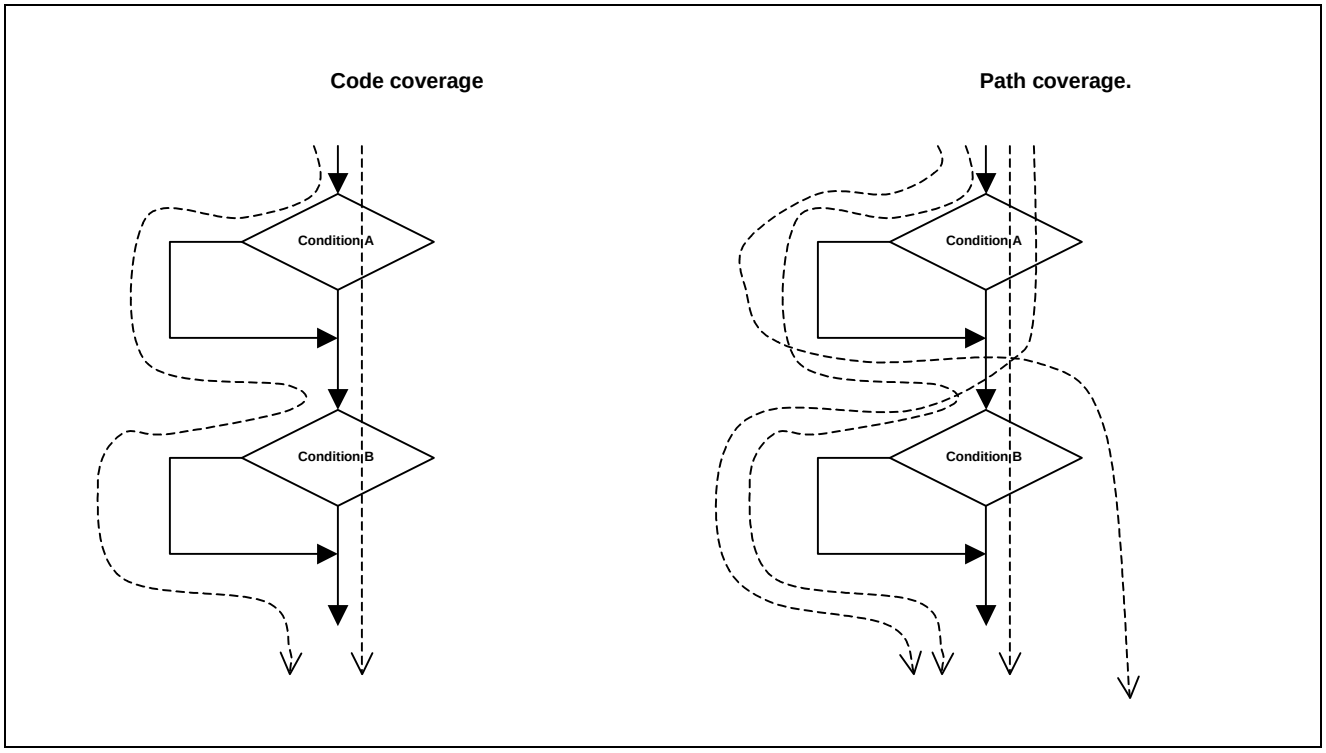
Path testing: execute all possible paths

Decision testing: execute all possible branches

Statement testing: execute all statements of the program

Generally the idea is to exercise all lines of code. If any of these codes are not executed, there is a greater possibility that a bug is not detected! There is no clear indication on how much coverage testing is required or enough. Ideally coverage shall be 100%. But this is difficult to achieve. If a guide is made to have a basic coverage of 85%, all testers will effectively stop working when this limit is reached! A general guide shall be a full testing of high-risk code (yet there is no clear rule!)

Example:



5. Debugging Techniques

Debugging is unavoidable and it will take up unpredictable time. Developers shall have good planning and practice to reduce debugging time. Developers approach in dealing with bug is important. A clear and fresh mind set will enable debugging work in a more efficient way.

The following are the theoretical approaches:

1. Changes cause bugs – Look for the difference.
2. Debug now instead of delaying the work.
3. Same mistake may happen twice- Track it down.
4. Read and analyze before any changes.
5. Explain the bugs to a third person – Reasoning and being questioned will bring new insight.
6. A reproducible bug is easier to solve.
7. Narrow down and segment the module to track the bugs.
8. Study the bug symptoms and output pattern.
9. Debugging at wrong area?
10. Other system bug – Compiler and emulator (but a number of developers are using them at the same time!)
11. Understand the debugging tool used.

The following are some techniques used to prevent and trap bugs.

1. Follow a good Coding standard.
   Developers must have good practice, as to how the C & header file is organized, definition used, …
2. Program for portability, re-usability and clarity
   A function should have a clear input & output parameters. This will make it easy to understand & portable.
   Constants used must be declared and remarked clearly in header files, instead of embedded within the functions. This will cause changes to be difficult.
3. Program defensively.
   Includes cases whereby it is not possible to happen, such as “other” in case statement, check for null pointers, …
4. Plan for error handling.
   Create timeout error if the waiting time is too long, return error code, …
5. Plan for debugging and testing
   Create debugging conditions and separately handle these conditions in the program. This will enable easy testing, such as to isolate hardware bug. The following are self-explanatory conditional definition.

```
#Define DEBUG              TRUE
#Define HARDWARE_READY     FALSE
#Define HOST_READY         FALSE
#Define SELF_TEST   TRUE
```

6. Define variable correctly (unsigned / signed integer long)
   Ensure all function prototypes and parameters passing are handling the same type of variables.
7. Place all components in a known initialed state when startup.
   — Are all variables, hardware registers, and data area initialized?
   — A bug may not be reproducible if the initial condition is different.
8. Place meaningful and correct remarks.
   — Incorrect remarks may mislead the reader.
   — The programmer name and date of changes are important information too.
9. Plan for the use of debugging tool.
   — An emulator user cable will need extra space.
   — A “JTAG-like” emulator may not have enough trace depth to track the bug. Thus software implementation of trace may be needed.
10. Plan for monitor/ debugging code.
    — A monitor code may be embedded into the application to perform basic debugging. Limitations of the monitor code have to be catered for at the initial stage.
    — Other debugging code may need to be prepared, such as blinking LED, printf to serial port/ memory space.
11. Prepare test pins and pads for testing - clock, ground Vcc, IRQ, external trigger…
    — Power and clock pins are essential test points to ensure proper operating condition.
    — Specific test points may be needed if current measurement is required.
    — It is wise to create extra pads to the unused pins of MCU or FPGA. These will enable easy re-wiring or probing.
12. Jumper designed
    — Silk-screen labeling will enable easy understanding of jumpers or switches setting.
    — However it is a good practice to consider:
      - Placement for easy access.
      - Switches to a “ON” state normally signify “Short-circuit”, “HI”, “enabled”
      - Jumper may fall off during transportation. This shall be the default operating state of the system. (e.g. Jumper-in to signify debug mode)
13. Mark (silk screen) pins number, signal and jumper name
    — Essential points shall have the signal name printed (e.g GND).
    — To enable easy probing, each surface mounted IC shall have a mark at an interval of 5 pins.

14. Has a power LED
    - It is quite common that troubleshooting end up discovering that the power is not connected.
    - A flicker of the LED may signify that the power line is being interrupted.
15. Do not suppress and ignore compiler warning.
    Each warning indicates a non-conformance. Execution may be “normal” at an instance. But an exception may cause serious bug.
16. Save/restore session and scripting
    HEW provides a save /restore session and script execution facility. This is a good automatic tool to ensure similar “initial “ state.
    If the sequence of setting is done manually, any step may be missed and the system behavior may be different.
17. Practice of estimation instead of assumption.
    Make basic calculation about transfer rate, CPU speed…the estimation ensure proper execution of tasks.
18. Echo input data
    It is a good practice to echo data received from another peripheral/host. Interpretation of data may be different/misunderstood. A wrong input will definitely give a wrong output!
19. DMA or interrupt interruption
    Interrupting events may disrupt activity that has to conform to strict timing. A DMA activity may cause the sequence of write to an I^2C device to fail. Programmers have to pay specific attention to timing sensitive events.
20. Fill unused area with a particular code
    - Fill unused interrupt vector to point to a debug routine.
    - Fill unused area with a BSR instruction to a debug routine.
      If a break is asserted in the debug routine, the fault can be detected immediately. Moreover the last executing location can be identified, by checking on the stack area. (The trace window of the emulator will provide more information)
21. Fill unused area with a predefined data
    Fill data, stack, or heap area with a data pattern of H’1234 or H’5A5A or H’A0A0. If a pointer obtained a data value of such pattern, an obvious pointer usage problem is identified. An even value is preferred, as this data may be used as an address pointer.
    If a regression test is done, developers can observe the utilization of the stack and heap area.
22. Handling for Analog Signal
    The hardware has to be handled carefully in term of treating the grounding and the possibility of digital interference.
    The static test of the hardware circuitry may prove the correct capturing of analog signal. However the reading of this signal through the ADC may not be correct dynamically. Thus it is wise to echo the capture data to a memory area, or pump a series of simulated data to the system. This will help to isolate the cause of error.
23. Counting interrupts
    A hardware counter can be used to probe the interrupt pin. All interrupt to the MCU will also increment the external counter. This enable programmer in determining the numbers of interrupt missed when the system is running at full speed.
24. Self test / diagnostics program
    There can be three different classifications of program/ routine for testing:
    A. A less complex program than the actual application (e.g. RTOS project). It should be written to test/verify the basic platform operation. This is to isolate possible issues arose from the initial hardware design and prototyping.
    B. A self-test routine that will activate and verify the operation of the standard application routines. This is to ensure and maintain the correct operation and characteristic of each routine. This self-test routine are defined within the system, so as to isolate possible issues deprive from communicating to the external devices.
    C. A PC based test script or program written to test the system.

25. Watchdog

If watchdog is used by the application to recover from disasters, it may cause more problems while troubleshooting. It may be wise to switch it off at development stage. If the watchdog is implemented with an external chipset, facility to switch off (Jumper) the function must be catered for.

However this watchdog function can be modified to facilitate debugging. Instead of allowing the activated watchdog to start execution from reset, software check within the reset routine can be done, so as to branch the routine to a debugging routine.

26. Use of Printf and Assert

Assert() is similar to a condition printf() statement.

Since embedded system do not have a standard output console, Developers has to plan for this functions. The debugging information can be output to a LCD or PC hyper terminal. However this method will take up significant CPU time. Alternatively the printf() function can be written to a RAM area. (Refer to Software Trace)

27. Software trace

Printf() is a good function to output relevant debugging data. If serial port is used as the output media, significant time is used. This may create real time issues. However using memory access will reduce the intrusiveness of this mechanism.

Operational notes:

— A customer-made printf function writing fixed length of data to a circular buffer, the buffer pointer will be incremental and reload to the beginning of the buffer upon reaching the end.
— Programmers insert printf function in all possible evaluation points.
— Upon “exit of program”, programmers are able to retrieve the “traced data” from the buffer through a command via the serial port.
— The buffer has to be preserved upon program crashed.
— The startup routine must be catered not to clear the buffer & pointer.
— Significant RAM may be used depending on intended trace depth.

28. Effective Usage of Development Tool

Make full use of the tool features such as

— Complex break system to trap a condition for evaluation.
— Guard access to limit program runaway.
— Events Trace to monitor program flow.
— Run time measurement to measure CPU utilization.
— Parallel-on-the-fly (POTF) to observe immediate effect when data is changed
— Step In/Out/Over to monitor code coverage manually.
— Memory/ file compare to evaluate data integrity, or external target working condition

Please refer to Application Note on
” Effective Usage of Hardware Development Tool”

29. Use of HEW utilities

Map Viewer	- To verify the generated code location in MCU map
Profile and Performance analysis	- To identify the program hot spot, so as to perform optimization.
Call Walker (Stack Analysis)	- To analyze the depth of stack required
Coverage	- To examine the coverage of the program
TCL/TK	- To use the script for consistent execution of operations.

Please refer to Application Note on
” Effective HEW Usage of Map Viewer”
“ Stack analysis using Call walker”
“Effective HEW Usage of Profile and Performance”
“Effective Usage of HEW Coverage”

30. Bug management

Bugs count is another aspect of bug control.

Developers shall track all bugs: Bugs must be counted and documented.

This process will lead to fewer bugs, and thus faster deliveries.

6. Hardware Troubleshooting Guide

The following are some key points to consider when performing hardware troubleshooting. For any new prototypes, hardware issues are not uncommon. Unless the prototype has gone through quality design verification, there is still a possibility that it may breakdown after long hours of (mis) usage.

6.1 Inspection

Visual check is as important as code walkthrough.

The focus does not fall on short and open circuit, but also on re-verification of schematic and specification.

A magnifier or a vision system will enable a better examination on cold solder, PCB crack line, solder bridge, wrong component, wrong orientation, etc.

6.2 Power Supply Check

The check does not mean a voltmeter verification of VCC. A physical scope monitoring of startup till operation is necessary. Any dip or ripple may introduce unwanted noise to the whole system. Capacitor (reservoir) of tenth or hundredth of farad may be required. Make sure each Vcc pin of the IC has a 0.1-uF decoupling capacitor placed closely to them.

$$C = \frac{I \text{ (required current drive)} \quad x \quad t \text{ (transition time)}}{V \text{ (tolerable ripple)}}$$

6.3 Clock Check

A scope must be used to verify the clock's frequency, rise time and fall time. This basic characteristic must be checked as some other capacitive or inductive element may cause the crystal to oscillate at another harmonics.

6.4 Reset Check

The reset condition must be met in accordance to the hardware manual specification. This is to ensure a correct initialization of the system. If capacitive load of the system is high, the power supply rise time may be delay, thus causing improper reset.

6.5 Timing

The hardware timing issues will arise if external memory or peripheral is being interfaced. The essential check point is the AC characteristic of both the MCU and external devices.

6.6 Power Up / Reset Sequence/ State

The power up and reset sequence must be carefully analyzed. The following are queries that may lead to the root issues.

- Is power up reset fed directly to all systems? Any power up sequence?
- Does the main MCU control the reset of the other sub-system?
- Is there any dual power supply? (Start up contention?)
- Will the startup state cause any conflict / contention?
- How much current is drawn during the transition?
- Will the reset state cause any risk to other systems? Need a Pull-up/down resistor?
- What is the state of the MCU or IC pins at the reset state? High impedance?
- Does the initial reset state require much higher current drive? Can the regulator supply the additional current?
- Is the system designed to be hot-plug?
- Are protection schemes such as reverse supply and high supply limitation incorporated in the system?

6.7 IC Drive & Interface

- Do all interfaces comply with the correct logic? (DC characteristic)
- Designed for 3V & 5V systems?
- Designed CMOS & Bi-Bolar systems?
- Is the input 5V tolerant?
- Is an output driving too many inputs? (Current drive may be sufficient, however the capacitive load may slow down the signal)
- Is the correct value used for the pull-up resistors?

6.8 Heat Run

Has the hardware system gone through the preliminary heat run test before embarking on the more complex software debugging?

A simple heat run can be a mere execution of self-test over a 24 hours period, or over a temperature chamber.

6.9 Coupling

- Is the PCB designed to withstand any noisy interference?
- Is there any high frequency and low frequency isolation?
- Is there any digital and analog isolation?
- Have the differential signals been handled specially?
- Are there enough grounding /return paths consideration?

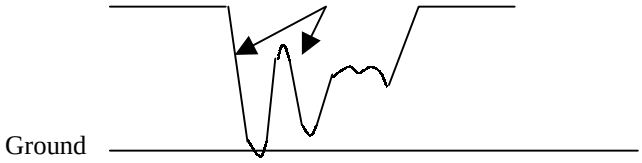
6.10 Signal measurement

The measuring device must be carefully selected. The logic analyzer, oscilloscope, multi-meter and probes used must be able to capture the wanted signal characteristic.

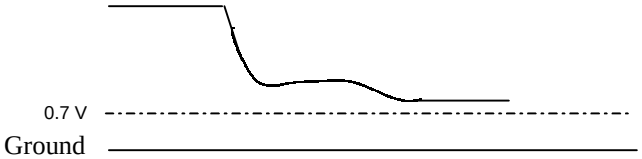
- A good logic analyzer can capture up to 4 ns of resolution. Thus verification of timing compliance must cater to the possibilities of error of the instrument.
- If a 100MHz oscilloscope is used to measure the operation of 50MHz, the real time error will not be captured correctly. A 500MHz or 1GHz scope may be needed.
- The Probe used is also very important. Noise measurement of pins required active probe of GHz range and has short probing & grounding lead.

Example of possible captured error:

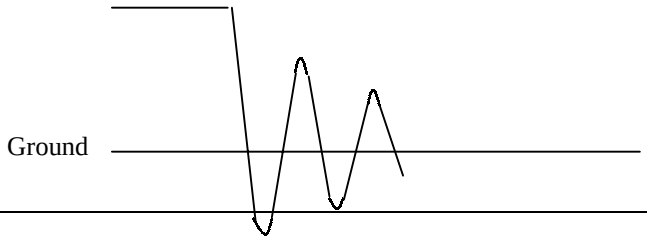
- Double edge triggers



- Floating signal.



- Noisy signal. (Reflection)



7. Summary

This document provides:

- Summary of bug types
- Pointers during design stage
- Different stages of test, which consist of static & dynamic testing methods on each unit, module and system.
- Highlight the theoretical & practical techniques in debugging
- Illustrated various hardware-troubleshooting guide.

Good planning and practice is the ultimate effective approach in dealing with bugs.

8. References

1. www.embedded.com
2. www.ganssle.com
3. www.testing.com
4. www.ednmag.com
5. “The Practice of Programming” by Brian W. Kernighan, Rob Pike, Addison-Wesley
6. “Writing Solid Code” by Steve Maguire, Microsoft

Revision Record

		Description	
Rev.	Date	Page	Summary
1.00	Sep.10.04	—	First edition issued

Keep safety first in your circuit designs!

1. Renesas Technology Corp. puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
   Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corp. or a third party.
2. Renesas Technology Corp. assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors.
   Renesas Technology Corp. assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corp. by various means, including the Renesas Technology Corp. Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corp. is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corp. for further details on these materials or the products contained therein.