Raptor Engineering Test Solutions

Raptor Engineering Automated Coreboot Test Stand (REACTS™)

User's Guide
For System Configuration
and Setup

Table of Contents

Chapter 1 Introduction to the Automated Test Stand

The Raptor Engineering Automated Coreboot Test Stand (REACTS™) is a combination of software and commercial off-the-shelf (COTS) hardware that, when combined, enable automated verification of coreboot functionality for a set of user-provided devices under test (DUTs). The verification software supports testing of both mainline GIT and Gerrit proposed changes; each test source can be activated or deactivated as desired. All tests run in parallel, with final verification being uploaded to Gerrit or the board status repository/mailing list once all tests have completed and results are known. The test stand supports automatic detection and recovery from DUT hardware failure; therefore, it is unlikely to generate spurious false negatives.

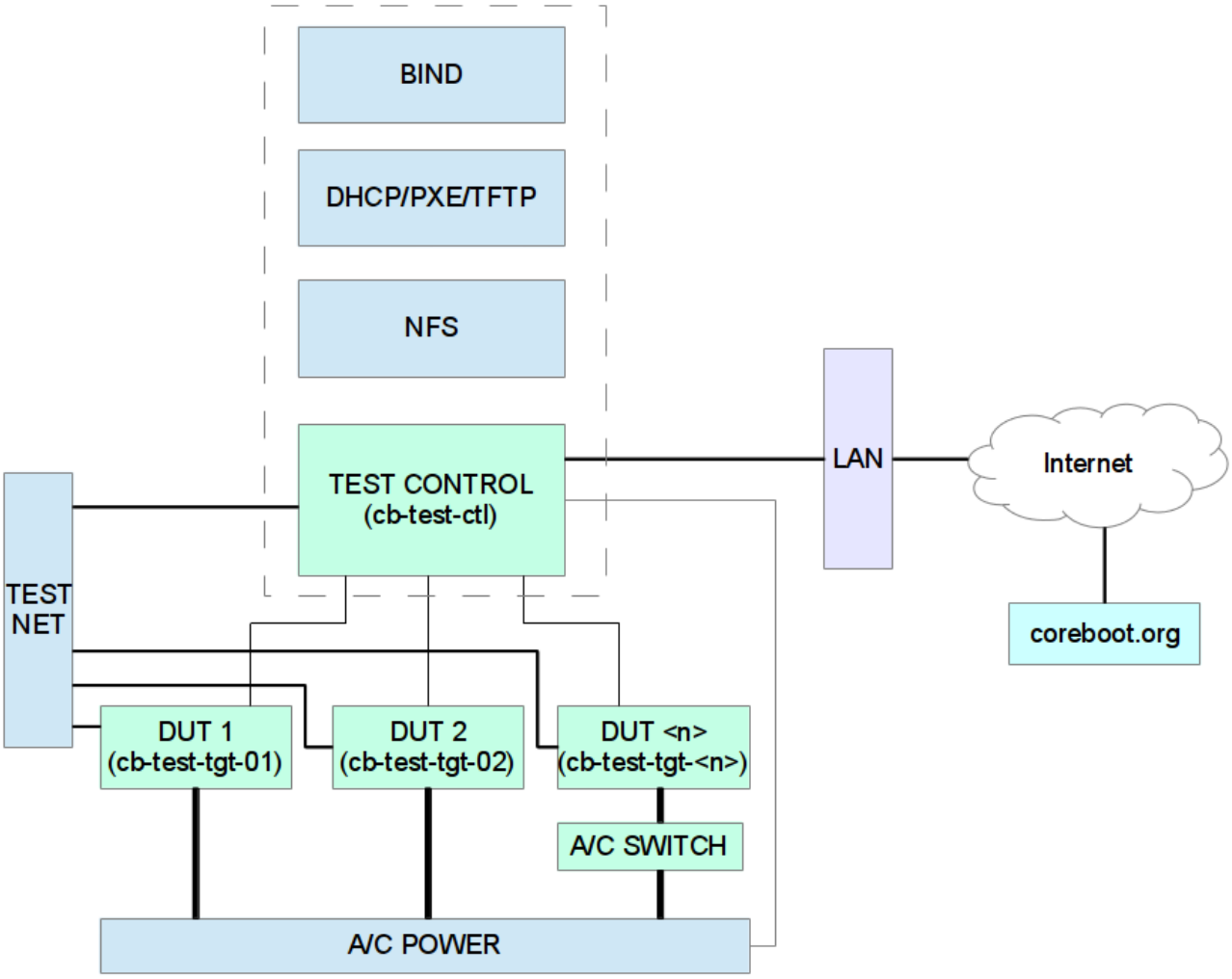


Fig. 1.1 Simplified Diagram of Automated Test Stand

To achieve this functionality, several systems are utilized. At the hardware level, a small, low-power, ARM-based single-board computer (SBC), such as a Raspberry Pi or a Beaglebone Black, is used to control device power, reset, and NVRAM clear for each DUT. Optionally, a USB-based main power switch may be utilized to forcibly cycle power to each DUT; this is useful for laptops or other systems with an embedded controller (EC) that may become unresponsive to the normal GPIO-based

power/reset signals.

Each DUT boots Linux from the central SBC. Linux is used for extended verification of DUT functionality; if Linux fails to boot, the DUT is immediately marked as failed. The verification process is controlled by the ARM SBC, and checks that:

1.) the DUT's CPU, RAM, and network adapter are functional
2.) coreboot's ACPI tables are valid
3.) DMI data is accessible
4.) cbmem is accessible and valid
5.) where enabled, nvram is accessible and operating normally

In the event of a DUT test failure, coreboot's fallback mechanism is utilized to boot a known-good romstage, ramstage, and payload. Provided the DUT's hardware is functional, coreboot is recompiled with serial debug enabled, and a second attempt is made to test the coreboot image. Log data is captured via the RS-232 serial connection during this second attempt and is stored for later upload and analysis.

The test process for QEMU is slightly different. As QEMU does not support nvram, a basic test sequence is used which does not utilize or test the nvram. Also, because QEMU requires a host computer, a dedicated server to which the test stand has ssh access is required.

All test control processes run on the SBC to save electrical power between test runs. Each mainline test run is fired via cron; when fired, the mainline test request is appended to the existing test queue.

Chapter 2 Basic Hardware Setup

All DUTs and their associated hardware / software connections must support the following tasks:
1.) Power on
2.) Forcibly power off
3.) Force fallback boot
4.) Flash new BIOS image / switch to new BIOS image

DUTs may optionally support the following tasks:
1.) Reset
2.) Clear NVRAM

For most physical devices, power on will be handled via a single GPIO line connected to the power switch. Most desktop and server mainboards (excluding those with a BMC that shares the system Flash memory) will be able to use that same line to forcibly power off via the 4-second hard-off feature built in to most SuperIO devices. Devices that do not have an independent SuperIO or BMC device (such as some high-end servers and all laptops) will need to use an external AC power control device to forcibly remove power from the DUT. QEMU support, instead, utilizes host secure shell access to start and stop the virtual machine(s) (VMs) as needed.

If the 4-second hard-off feature is used, it is recommended that the reset line be connected to a GPIO so that coreboot does not attempt to start during the force-off procedure.

For most physical devices, flashrom, running under Linux, will be used to flash a new BIOS image; however, for QEMU, the new BIOS image file will be uploaded to the appropriate location, and the QEMU start command will use the new BIOS image file when requested.

In order to support a wide variety of DUTs, a simple Python script (`board_integration/target_control`) is used to execute the specific commands required to perform the above listed functions. This script calls functions in the user editable file `/opt/raptor/dut_control.py`; this file must contain, for each connected DUT, appropriate code for the following commands:

```
power_on_normal
power_on_fallback
force_power_off
resume
flash_rom
```

Each control function makes two parameters available to the Python code contained therein, `target_number` and `target_hostname`. These parameters correspond to the DUT configuration variables set in the main `reacts.conf` file, and are intended to make DUT control easier for certain common tasks.

The provided sample control file includes sample code for both a server mainboard with independent SuperIO and for QEMU. Recommended wiring, signal sequences, and timings for SuperIO-driven boards may be found in Appendix B.

Each DUT requires a dedicated RS-232 serial connection for capturing coreboot logs in the event that normal access mechanisms fail. It is recommended that one specific model of USB to serial converter be utilized, and that the requisite converters are attached to the control system through no more than

one level of USB hub. If these conditions are met, the REACTS™ system assigns each serial port a unique device node based on its physical location on the USB hub(s); these device nodes start with `/dev/ttyCBTGT` and end with a unique two-digit number.

All DUTs are connected to an isolated, REACTS™-internal network. The Ethernet port on the DUT connects to the external, Internet-connected network, while the internal network is serviced by a dedicated USB to Gigabit Ethernet adapter.

Certain targets, for example those with SPI-based Flash devices, can use an external flash mechanism. Each REACTS™ control system contains an SPI master that can be used to externally write firmware images to the SPI Flash memory of a single target. This method of operation requires that the target fully release the SPI control signals before the external firmware write can proceed; the recommended wiring shown in Appendix B assumes the Flash device has been completely removed from the DUT and is now present on the interface board. The recommended wiring also forcibly disconnects the associated signals from the target before attempting external SPI access.

On certain targets this ROM removal and forcible disconnection may not be necessary to support the external flash mechanism. If forcible disconnection is not used, please note that some target board designs may require that the target be held in reset while target power is applied, while other targets may allow SPI access with target power removed entirely. Please see Appendix B for further wiring information.

Chapter 3 Software Configuration

3.1 Initial Setup and Login

After extracting the REACTS™ firmware image to your SD card and installing the firmware card into your SBC, you are ready to boot the system and run through initial setup and configuration tasks.

NOTE: When first starting your REACTS™ system, note that a configuration dialog may be displayed with several options including one to expand the root filesystem. DO NOT choose this option as it will have an undefined / undesirable effect upon the REACTS™ system.

The default login for the SBC is:
```
username: pi
password: raspberry
```

All of the setup commands in this document assume that you are the root user on the REACTS™ system. Therefore, you should switch to a root shell after login by executing:

```
sudo bash
```

Before using the REACTS™, or if you need to restore the default DUT system files for any reason, execute the following command to unpack the DUT system files:

```
/coreboot/create_target_nfsroots initial
```

After unpacking the DUT system files, you will need to restart the REACTS™ service before executing any tests.

3.2 Configuration

The test stand configuration is stored in a single file on the control system (`/opt/raptor/reacts.conf`). The file has the following syntax:

```
[global]
board_count = 2
usb_serial_vendor_id = 067b
usb_serial_product_id = 2303
external_dns_server = 8.8.8.8
external_ntp_server = 0.pool.ntp.org
automaster_interval = 1
test_result_base_url = https://myhost.com/coreboot/autotest/logs/
owner_email = <your Email address>
toolchain_update_command =

[smtp]
host = <your SMTP host>
user = <your SMTP username>
pass = <your SMTP password>
target = coreboot@coreboot.org
report_build_failures = 0
```

```
min_interval = 86400

[gerrit]
port = 29418
host = review.coreboot.org
user = <gerrit user>
email = <test stand commit Email address>
author = Raptor Engineering Automated Coreboot Test Stand
pkey = /root/.ssh/id_rsa_gerrit
mode = reviewer
project = coreboot
branches = master

[board 0]
enabled = 1
extended_test = 1
video_test = 0
suspend_test = 0
external_flash = 0
external_flash_failsafe_rom =
friendly_name = ASUS KFSN4-DRE
serial_device = /dev/ttyCBTGT01
serial_remote_host =
serial_remote_pkey =
camera_device =
host_name = cb-test-tgt-1
boot_timeout = 300
build_timeout = 3600
video_decode_timeout = 10
architecture = x86_64
net_adapter_devname = eth0
kernel_boot_arguments =
ipxe_rom_vendor = 14e4
ipxe_rom_device = 1659
mac_address = 00:11:22:33:44:55

[board 1]
enabled = 1
extended_test = 0
video_test = 0
suspend_test = 0
external_flash = 0
external_flash_failsafe_rom =
friendly_name = QEMU x86_64 Q35
serial_device = /dev/ttyS11
serial_remote_host = <QEMU host>
serial_remote_pkey = <QEMU host SSH keyfile>
camera_device =
host_name = cb-test-tgt-2
```

```
boot_timeout = 300
build_timeout = 3600
video_decode_timeout = 10
architecture = x86_64
net_adapter_devname = eth0
kernel_boot_arguments =
ipxe_rom_vendor = 8086
ipxe_rom_device = 100e
mac_address = 80:80:80:80:80:ff
```

Three main section types are supported:

global

Contains global information such as the total DUT count, USB to serial adapter type, and external DNS / NTP server addresses. `automaster_interval` specifies the minimum delay between test runs against GIT master in hours; setting this to zero will disable testing of the GIT master branch.

smtp

Configures the external SMTP server used to report test failures. To disable external Email error reporting, set the `host` field to an empty string. `min_interval` sets the minimum number of seconds that must elapse before another failure message is sent.

gerrit

Stores host, port, project, and authentication information for the master Gerrit server. The `email` and `author` fields are not used to authenticate to Gerrit; they are only used to create author information for the board-status repository. The REACTS can be also configured to test multiple branches; to use this feature, simply provide a comma-separated list of branches to test in the `branches` variable.

board <n>

Stores DUT-specific information, including the serial device to which the current DUT is connected. If the current DUT requires any Linux kernel options to boot correctly, these options should be provided in the `kernel_boot_arguments` field. Similarly, if the network adapter connecting the DUT to the REACTS™ control system is not the default `eth0` under a Linux kernel, the correct network adapter device name should be provided in the `net_adapter_devname` field.

The gerrit configuration section should not need to be modified except to enter your Gerrit bot account username, key file location, and to set the testing mode. Two testing modes are supported, `reviewer` and `all`. Reviewer mode only tests changesets for which the REACTS™ has been added as a reviewer, while All mode tests all changesets that are uploaded to or modified on Gerrit.

The board_count variable in the global section sets the number of DUTs connected to the control system; there must be an identical number of monotonically incrementing [board n] sections provided within the configuration file.

The REACTS™ uses a prebuilt toolchain to build test images in order to provide usable testing rates. When changes are made to the coreboot sources that alter the toolchain subcomponent versions, builds may fail due to the stale, cached toolchain binaries. The REACTS™ detects this condition in the regularly scheduled automaster builds, and will subsequently execute any command provided in the

`toolchain_update_command` configuration directive. If no command is present, or if the provided command exits with a non-zero return code, the REACTS will leave the toolchain failure flag set and abort all subsequent tests pending a required manual toolchain update. Licensed REACTS™ users with an active support contract have access to signed build images generated by Raptor Engineering; for these users, the REACTS™ will automatically download and use the latest toolchain image on detection of a toolchain failure.

Each board can be disabled manually by setting the enabled field to 0. When a board is disabled, it is not included in the autotest results. Extended test is appropriate for most devices and should be set unless nvram support is not available for the selected DUT. The friendly_name field will be used to label the DUT in all public autotest result entries, including Gerrit verification posts.

Each DUT must have its `architecture`, serial_device, boot_timeout, ipxe_rom_*, and mac_address variables set appropriately. For most DUTs, a boot_timeout of 300 seconds is adequate; however, on slow DUTs or server boards with large amounts of ECC RAM, this timeout may need to be increased. Any DUT that has not fully booted into Linux before the timeout expires will be assumed to have failed verification.

After editing the configuration file, depending on which settings have been changed, you may need to restart the REACTS™ service before the new settings will take effect. In general, settings in the `board` sections do not require a restart, while changes in the `global` and `gerrit` sections do require a restart. However, if any board `architecture` settings are changed you will need to update the boot kernel images and DUT filesystems by executing `/coreboot/create_target_nfsroots`.

The REACTS™ can make use of an external NFS server via the optional `external_nfs_server` directive, provided that certain server requirements are met. If you require this option for your test environment, please contact Raptor Engineering at support@raptorengineering.com for more details.

3.3 Remote Logging

Certain systems, such as QEMU and various laptops, do not have a physical serial port available for capturing debug logs. Many of these systems provide an alternate means for log reception, for example USB EHCI debug via a host system. The REACTS™ can capture debug logs from a device node on a remote system; to use this feature the REACTS™ must have the ability to use an SSH private key to log in to the debug host system. Remote log capture is controlled by the following two lines in the `board <n>` section of the configuration file:

```
serial_remote_host
serial_remote_pkey
```

If `serial_remote_host` is not blank, the REACTS™ will attempt to log in to that system as root with the SSH keyfile specified in `serial_remote_pkey`, and capture logging information from the device node specified in `serial_device`. No setup is performed on the device node; debug information is expected to be made available from the device node via a simple `cat` command.

3.4 PCI Device Check

Each DUT can be scanned to verify that all specified PCI devices have been properly enumerated after boot has completed. To use this option, simply create a file in `/opt/raptor/` following this naming convention:
`reacts_board_<board_id>.pci`

In the `.pci` file for each DUT, list all PCI vendor and device IDs for which you wish to verify presence. Order does not matter, but if a device is listed more than once the REACTS™ will test for the presence of the total number of devices listed. For instance, a file named `reacts_board_0.pci` and containing the following entries will test for the presence of two Broadcom® NIC devices and an XGI® Volari™ device on DUT 0:

```
18ca 0020
14e4 1659
14e4 1659
```

If a device listed in the assoicated `.pci` file is not present, the REACTS™ will report a PCI device enumeration failure. If devices are present that are not listed in the associated `.pci` file, the REACTS™ will ignore them and proceed with the rest of the testing sequence.

3.5 Video Usability Test

The REACTS™ supports validation of display device functionality for each DUT. To use this feature, you will need a separate USB webcam attached to the REACTS™ control system for each DUT with enabled display validation. Ensure that each webcam can see the entire display of the associated DUT and that the screen is in focus, for example by using `zbarcam` in graphical mode. Display validation is controlled by the following three lines in the `board <n>` section of the configuration file:

```
video_test
camera_device
video_decode_timeout
```

If `video_test` is not 0, the REACTS™ will attempt to use the video device specified in `camera_device` to verify the functionality of the DUT display. The REACTS™ will wait up to `video_decode_timeout` seconds after DUT boot for display validation to succeed. If the display device attached to the DUT has an unusually long warm up period this value may need to be increased, otherwise the default of 10 seconds should be adequate.

If the webcam associated with a DUT is subsequently moved so that it cannot see the entire display of the DUT, or so that the focus is insufficient to obtain a clear picture, display validation will fail. As such, it is important to ensure that the webcam is affixed to a reasonably immobile object on order to avoid false negative test results.

3.6 Suspend Operation Test

Correct operation of S3 (Suspend to RAM) can be tested by the REACTS™, provided that hardware and firmware support for the S3 mode exists on the DUT, and that the Linux kernel is capable of utilizing this support. To enable testing of the DUT's S3 suspend mode, first ensure that `resume` support for your DUT has been added to `/opt/raptor/dut_control.py`, then enable testing via

the suspend_test option in the board <n> section of the configuration file:

```
suspend_test = 1
```

3.7 External SPI Firmware Write

The REACTS™ natively supports writing firmware images directly to a single DUT over SPI. Other protocols and additional SPI DUTs are also supported provided that compatible external hardware adapter(s) are attached to the REACTS™ control system. To enable the external firmware write feature, ensure that the physical wiring to the DUT's SPI Flash ROM is completed per the example in Appendix B, then enable external flash write via these two lines in the board <n> section of the configuration file:

```
external_flash = 1
external_flash_failsafe_rom = /coreboot/target_files/fallback_roms/<x>.rom
```

Note that you will need to transfer the built fallback ROM from the DUT to the location specified in external_flash_failsafe_rom prior to enabling the external firmware flash feature. Please see Chapter 4 for more information on building the fallback ROM image.

3.8 DUT Test Failure Reporting

By default, the REACTS™ system reports any test failures to the coreboot mailing list via the report_board_test_failure script, which uses the mutt program to assemble and send a failure message. The user-editable failure message body is stored in /opt/raptor/reacts_failure_message_body.inc. If transmission of failure messages to the coreboot mailing list is desired, you will need to configure the SMTP server via the smtp section in the REACTS™ configuration file, and subscribe the Email address used to report errors to the coreboot mailing list.

3.9 REACTS™ Service Control

The REACTS™ service may be restarted via:

```
/etc/init.d/reacts restart
```

It is not recommended to restart the REACTS™ service while any DUTs are booted and active.

3.10 DUT Malware Purge / Redeployment

Due to the high privilege level of the software and firmware being tested on the DUTs, there is a small possibility that a malicious changeset could install malware or other type of unwanted persistent software on one or more DUTs. While this would not affect the REACTS™ control system or QEMU host machine by design, it is not desireable to leave any such malware installed on the DUTs as they have some limited Internet connectivity and thus could be used for nefarious purposes.
To mitigate this risk, a new script has been introduced in REACTS™ system firmware version 1.2.0, /coreboot/redeploy_target_nfsroots. When executed, this script will wait for the current test cycle to complete (if any is active), then delete and redeploy the DUT nfsroots from the known-good tarballs present on the REACTS™ control system. All coreboot configuration data and fallback ROM images from the DUT are preserved. During execution of the cleaning script, all testing services are locked out to prevent spurious failures; when cleaning is complete queued tests will resume. As the

speed of the cleaning operation depends strongly on the performance of the disk subsystem on which the nfsroots are stored, it is left to the user to decide how often this cleaning operation should be run.

3.11 DUT Configuration Backup / Restore

If for any reason the REACTS™ control system needs to be reimaged, it may save considerable setup time to backup and restore the DUT configuration files. Two scripts have been provided for this purpose, `back_up_dut_configuration` and `restore_dut_configuration`. Both scripts take a backup file name as the only argument; after creating the DUT backup file, the following files should be transferred off of the REACTS™ control system onto an external storage device for restoration after reimaging has completed:

- The DUT backup file created via `/coreboot/back_up_dut_configuration`
- `/opt/raptor/reacts.conf`
- `/opt/raptor/dut_control.py`

All of the above mentioned files should be restored to the REACTS™ control system after reimaging, and `/coreboot/restore_dut_configuration` should be run with the DUT backup file created earlier to fully restore the DUT configuration files to the new system image.

Chapter 4 DUT Initial Setup

Each DUT requires a small amount of setup before it can be enabled. The typical setup process follows:

Install coreboot

Follow the board-specific instructions from the coreboot project to create a coreboot image with SeaBIOS payload, then flash it to your device. This coreboot image should include iPXE support for your network adapter. Verify that coreboot functions properly on your device before proceeding.

Connect the DUT

Physically connect the DUT to the REACTS™ test system. At minimum, you will need a network connection to the REACTS™ internal network, power control signals from the control system to the DUT (see Appendix B for recommended hookups), and a RS-232 null-modem serial connection between the DUT and the control system. If a laptop or other machine with an embedded controller (EC) is being tested, you likely will need to connect the DUT's power suppy to a REACTS™-controlled mains power switch in order to ensure the target is powered down when required by the control software.

Add the DUT to the configuration file

Add or modify a board entry in the REACTS™ configuration file for the new DUT. Verify that the MAC address is set correctly, and that the new DUT is disabled (`enable = 0`). Also ensure the DUT is connected to one of the serial ports on the control system, and that the device node for that serial port is set in the new DUT's board configuration section.

After modifying the REACTS™ configuration file, update the boot kernel images and DUT filesystems, then restart the REACTS™ service:

```
/coreboot/create_target_nfsroots
/etc/init.d/reacts restart
```

Set up coreboot

Connect the DUT to the internal REACTS™ network and boot it via iPXE. From the REACTS™ control system, execute the following command:

ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@cb-test-tgt-0<n>

where <n> is the board number previously configured in reacts.conf.

Once you have access to the new DUT, enter the following command:

```
/coreboot/build_coreboot_test_image menuconfig
```

You should now see the coreboot menuconfig screen. Configure coreboot for this DUT, then exit. Recommended settings are to enable CMOS support, switch to the normal bootblock (if possible), and enable a SPEW debug log level.

Create the fallback ROM image

After you have exited the coreboot menuconfig system, execute the following command:

```
/coreboot/build_coreboot_test_image fallback <manufacturer id> <device id>
```

where <manufacturer id> and <device id> are the Vendor ID and Product ID of your networking device, respectively.

Congratulations! The DUT is now configured for automatic test. Power off the DUT via

```
init 0
```

QEMU setup

QEMU testing relies on a dual-ported QEMU host machine, with one network port physically attached to the internal test network and the other network port attached to the same external LAN as the REACTS™ control system. The qemu guest needs to have the internal test network port attached, and the REACTS™ control system needs to have ssh access to the QEMU host over the external LAN in order to start and stop QEMU virtual machines. The user is responsible for configuring the QEMU host using bridged networking between the internal test network interface and any QEMU DUT virtual machines contained on the host.

Due to the lack of a BIOS update utility and physical serial port, QEMU DUTs require additional setup to function correctly. You may safely skip this step for non-QEMU DUTs. The goal of this process is to allow the QEMU DUT to make test ROMs available to the QEMU host, even after the QEMU DUT has been powered down, and to make a virtual serial cable available between the QEMU DUT and host.

First, set up an NFS export (`/coreboot/roms`) on the QEMU host. Allow the QEMU DUT to mount this directory read/write. Next, boot the QEMU DUT and log in from the REACTS control system via the previously given `ssh` command. Then, open `/etc/fstab` on the QEMU DUT for editing and append the following line:

```
cb-test-qemu-host:/coreboot/roms      /coreboot/roms  nfs   rw    0     0
```

Adjust the content if necessary to match your configuration, then save the file and exit the editor.

Before your QEMU DUT can be used, a virtual serial cable must be provided between the DUT and the host machine so that debugging information can be captured and reported if necessary. An example setup script has been provided in the following location:

```
/coreboot/qemu_host_files/scripts/start_serial_bridge
```

Copy the referenced script to the QEMU host and modify as needed, then configure the QEMU host to run that script once on every boot.

Finally, power off the DUT via

```
init 0
```

Enable automatic testing of the DUT

Set `enable = 1` in the main REACTS™ configuration file to enable automatic testing. You do not need to restart the REACTS™ service for this change to take effect.

Chapter 5 Gerrit Integration

The REACTS™ system supports automatic testing of proposed changsets uploaded to Gerrit. To use this feature, you will need a Gerrit bot account and valid SSH key.

To activate, first upload your Gerrit bot SSH key to the test control system – for example, to `/root/.ssh/id_rsa_gerrit`. Next, configure the `gerrit` section in the REACTS™ configuration file with your bot account information and key file location. Finally, restart the REACTS™ service.

If you wish to make the test logs available to the general public, you will need to set up a Web server that publishes the contents of the `/coreboot_build_logs` directory; then, set `test_result_base_url` to the base URL of that Web server. For example, if you make the coreboot test log directory available at http://myhost.com/coreboot, you will need to set `test_result_base_url = http://myhost.com/coreboot.`

To request testing of a changeset on Gerrit, add the aforementioned Gerrit bot account as a reviewer to that changeset. The REACTS™ system automatically tests the active changeset when added as a reviewer – and continues to test any changesets uploaded thereafter, posting test results to that Gerrit changeset.

Chapter 6 Test Flow

This section details the overall flow of a test, from remote change through local DUT boot and verification, to upload of test results to the remote server.

All test requests are queued inside the `monitor_gerrit` process. Two test request types are currently supported: Gerrit-initiated requests, and automaster requests. Automaster requests are fired via a standard cron job, and, unlike Gerrit requests, do not stack up in the queue. Gerrit requests are initiated when the Gerrit bot user associated with the test stand is added as a reviewer to a changeset, or when a re-review request is sent from a changeset on which the same Gerrit bot user already is set as a reviewer.

When a test request reaches the head of the queue, `execute_test_on_target` is called with appropriate command line parameters. This, in turn, calls `board_integration/target_control` with the appropriate parameters to power on the DUT(s) in fallback mode. As soon as each DUT is booted, `/coreboot/build_coreboot_test_image` is executed on that DUT; this script is responsible for fetching the specified changes from GIT and building the new test ROM image.

As soon as a DUT has finished building the test ROM image, `board_integration/target_control` is called again to flash the test image to the DUT's ROM. After the flash process completes, the DUT is fully powered down and rebooted once to ensure that faulty or changed NVRAM settings do not cause a false negative result. After power down, the corresponding serial port is opened and the DUT is powered on for testing. If the DUT boots into Linux before the prescribed timeout period has elapsed, several tests are executed within the Linux environment to check for the presence and correctness of cbmem data, nvram data, and other BIOS-provided or BIOS-controlled data structures. Provided all these tests are successful, the DUT is again powered down and rebooted for final analysis. If any of the various checks fail, including if the DUT does not boot, the DUT is forcibly powered off and rebooted in fallback mode. Once the DUT boots into fallback mode, a second build sequence is executed, this time with the serial console enabled. After this new debugging image has been flashed, the previous test sequence is repeated and the results are stored.

When running in automaster mode, each DUT independently either uploads a board-status report via the `/coreboot/upload_board_status_reports` script, or the test thread reports failure to the mailing list via the `report_board_test_failure` script. In Gerrit mode the main test script waits for all DUTs to finish testing, then agglomerates all results into a final verdict. In all cases, the RS-232 output from failed builds is saved to the `/coreboot_build_logs` directory on the REACTS™ control system.

Chapter 7 Troubleshooting

Symptom: The REACTS™ system consistently marks a single DUT as failed.

Resolution: There are several potential causes for this failure. First and foremost, verify that coreboot is fully functional on the DUT by checking for bootability, verifying full cbmem functionality, and verifying that nvram works. If the target boots but cbmem and/or nvramtool do not function – and cannot be repaired – you may want to set the `extended_test` key for that board to 0 in order to to bypass all checks except basic bootability.

Symptom: The REACTS™ system is unable to test any changes, and reports a GIT SCM failure.

Resolution: The most likely causes of this failure are a lack of network connectivity and an invalid REACTS™ Gerrit configuration. Try pinging the host specified in the `gerrit` configuration section to check for network connectivity. If that works, verify that the Gerrit username and SSH key are valid, that the SSH key has been installed into the specified location on the REACTS™ system, and that the correct Gerrit project name has been specified.

Symptom: DUTs randomly fail to pass testing, sometimes with strange symptoms, such as all extended tests failing or the resultant debug boot functioning normally.

Resolution: This may be caused by a timeout that has been set too low. Try increasing the timeout value to see if the system becomes stable. The timeout is a balance between decreased overall test time in the event of boot failure and false positives in the case of a transient slowdown (e.g. increased network traffic).

If you are encountering a technical difficulty that is not described here, please contact us at support@raptorengineering.com with a detailed description of your problem. We will do our best to assist with the issue. Entities holding a support contract will receive a response within one or two business days, with same day response turnaround being typical.

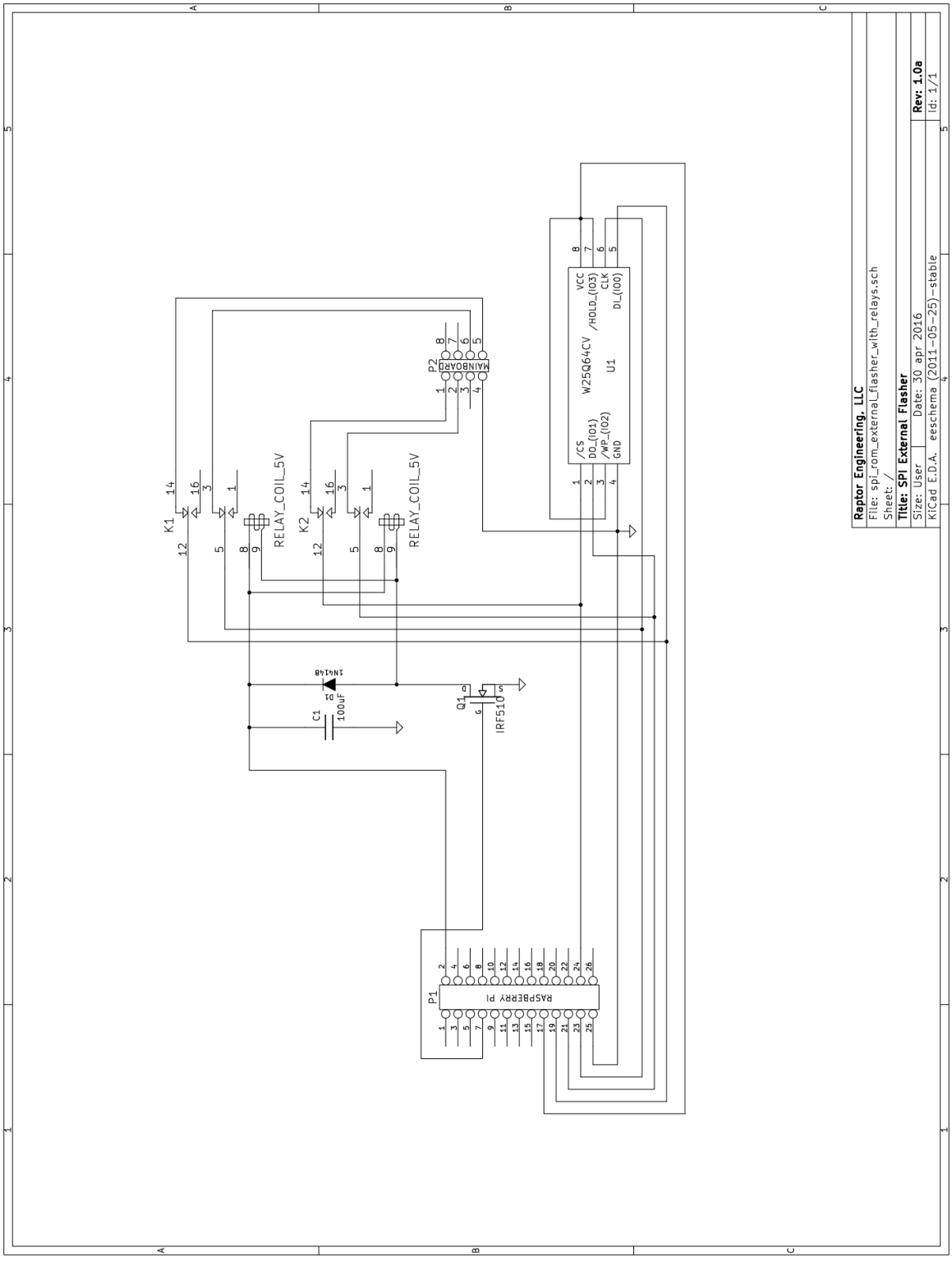
There are several log files generated by the REACTS™ system (`/var/log/coreboot_gerrit*`); these files may contain valuable information that could aid in troubleshooting. We ask that any requests for assistance contain a copy of these log files for further analysis.

Appendix A Required Hardware

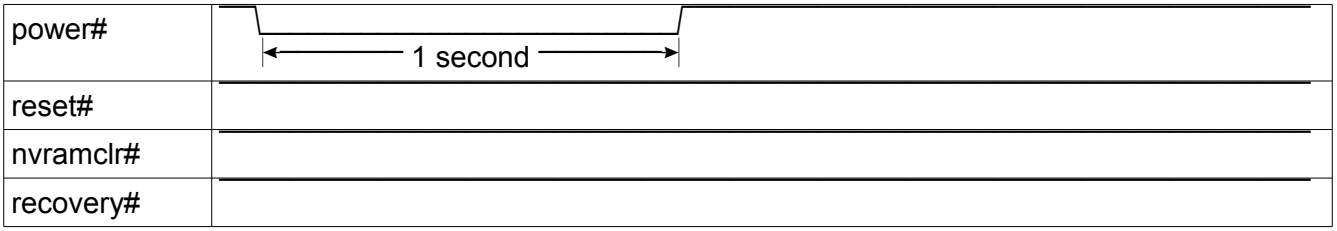
The following hardware is recommended for use in the REACTS™ system:

1 Raspberry Pi v2 (mandatory)
1 AX88178-based USB to Gigabit Ethernet adapter
1 5-port or 8-port Gigabit Ethernet switch (adjust according to the number of DUTs expected)
2 EnerGenie EG-PM2 programmable power strips (optional)
2 4-port USB hubs
5 FTDI USB to RS-232 serial adapters (adjust according to the number of DUTs expected)

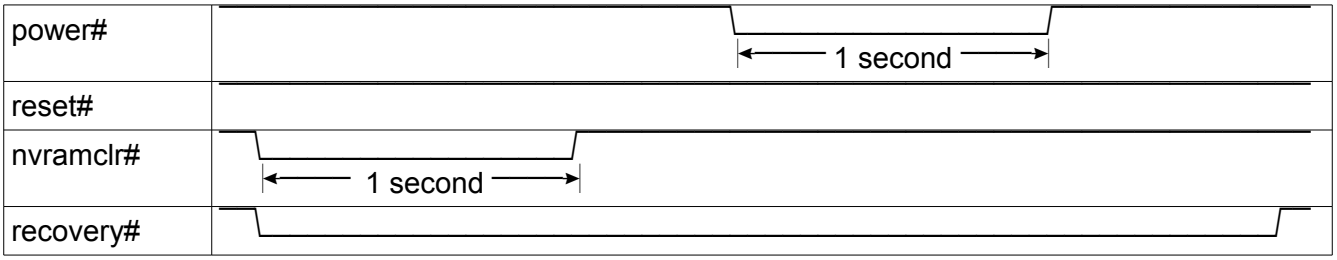
Appendix B DUT Signals and Wiring



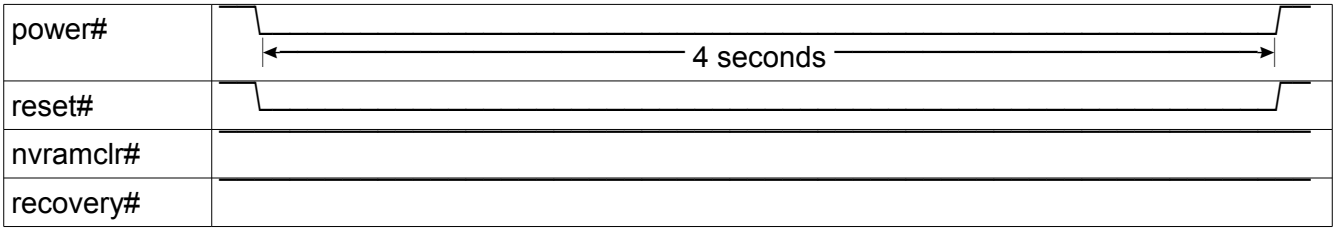
Recommended normal power-up sequence for Winbond/Nuvoton SuperIO devices:



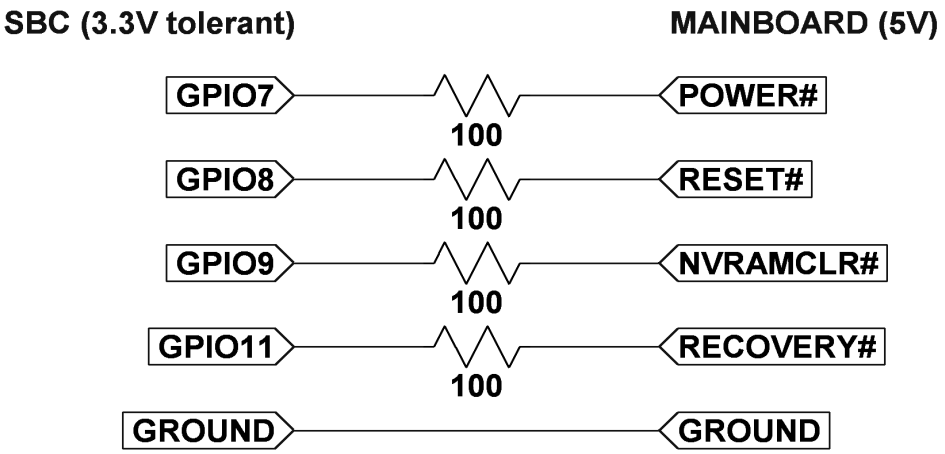
Recommended fallback power-up sequence for Winbond/Nuvoton SuperIO devices:



Recommended forcible power-down sequence for Winbond/Nuvoton SuperIO devices:



Recommended control wiring for a single SuperIO-based mainboard:



Recommended external SPI firmware connection method for targets with DIP-8 / SIP-8 Flash devices: