



# Automated Test Framework (FUEGO)

## for Linux System Test

### Jenkins-based Test Automation

Renesas Design Vietnam Co., Ltd  
System Verification Team  
R-Car Software Solution 3 Group  
Software Solution 2 Section

---

---

## **Revision History**

<b>No.</b>	<b>Date</b>	<b>Page</b>	<b>Contents</b>	<b>Author</b>	<b>Reviewer</b>
1.00	May 28, 2015	All	Create newly	RVC/Triet Huynh	RVC/Nguyen Nguyen
1.10	Aug 17, 2015	All	Revised based on review comments with REL	RVC/Triet Huynh	
1.20	Sept 04, 2015	3	Add information/guide for HW Reset function	RVC/Phuc Nguyen	
1.30	Sept 24, 2015	3	Revise part: 3. Hardware reset implementation for JTA (update algorithm)	RVC/Ngoc Nguyen RVC/Thu Nguyen	RVC/Nguyen Nguyen
1.40	Apr 21. 2016	3	Revise part: 3. Hardware reset implementation for JTA (update after creating detailed design)	RVC/Ngoc Nguyen	
1.50	June 7. 2016	All	Revise all for Salvator-X H3 board	RVC/Bao Hoang	
2.0	Jul 8. 2016	All	Migrate to FUEGO	RVC/Linh Phung	

---

# Contents

1	Introduction .....	6
1.1	Board Environment .....	6
1.2	Automated Test Framework .....	<b>Error! Bookmark not defined.</b>
1.3	Jenkins-based Test Automation .....	7
1.4	FUEGO Installation.....	8
1.4.1	Setup of Host PC .....	8
1.4.2	Setup of Target Board.....	11
1.4.3	Setup of Hardware reset board .....	14
1.5	FUEGO Structure .....	16
1.6	FUEGO Work Flow .....	18
2	FUEGO Implementation for Linux ST using SSH.....	19
2.1	Background .....	19
2.2	Modification for FUEGO .....	19
2.2.1	Overview .....	19
2.2.2	Block diagram .....	20
2.2.3	Modification diagram .....	21
2.2.4	Sequence Diagram.....	23
2.3	Detail Modification .....	24
2.3.1	Board configuration.....	24
2.3.2	Main script.....	25
2.3.3	Other scripts .....	25
2.3.4	Test Log Structure .....	28
2.3.5	Functions of Linux System Test .....	28
2.3.6	New parser.....	29
3	Hardware reset implementation for FUEGO.....	34
3.1	Background .....	34
3.2	Overview of Hardware reset control .....	35
3.2.1	Component of Hardware reset board.....	35
3.2.2	Reset activate sequence.....	36
3.3	Modification to implement Hardware reset to FUEGO .....	36
3.3.1	Block Diagram .....	36
3.3.2	Modification Diagram.....	39
3.3.3	Sequence Diagram.....	40
3.4	Detail modification.....	43
3.4.1	Main script.....	43
3.4.2	Other scripts .....	45
4	Example: Implement a test case .....	49

---

5	Restriction .....	56
6	Reference.....	57

# 1 Introduction

This document describes how to apply an Automated Test Framework (ATF) to Linux System Test (ST).

Fuego is the official Automated Test Framework for the LTSI project (LTSI Test Project)

It was formerly called "JTA" (for Jenkins-based Test Automation), but the name was changed in March 2016 for branding purposes.

The purpose of Fuego is to provide a test framework for testing embedded Linux, allow people to share their tests with each other.

This guide is used to implement Linux ST for R-Car board series on FUEGO framework. All examples in this document are based on R-CarH3 - Salvator-X board's architecture and configuration.

The Introduction part introduces overview about ATF and FUEGO Structure.

Note that readers should have basic knowledge of Fuego, programming languages such as Shell Script, Python.

## 1.1 Board Environment

To execute Linux ST on FUEGO, it requires Linux Host PC and some equipment to use Linux BSP on target board (R-CarH3 – Salvator-X). Please see below figure (Linux file system is on SD Card 0 – CN13). Also, please refer to Yocto Startup Guide “RENESAS\_RCH3M3\_YoctoStartupGuide\_UME\_v2.9.0.pdf” for more details.

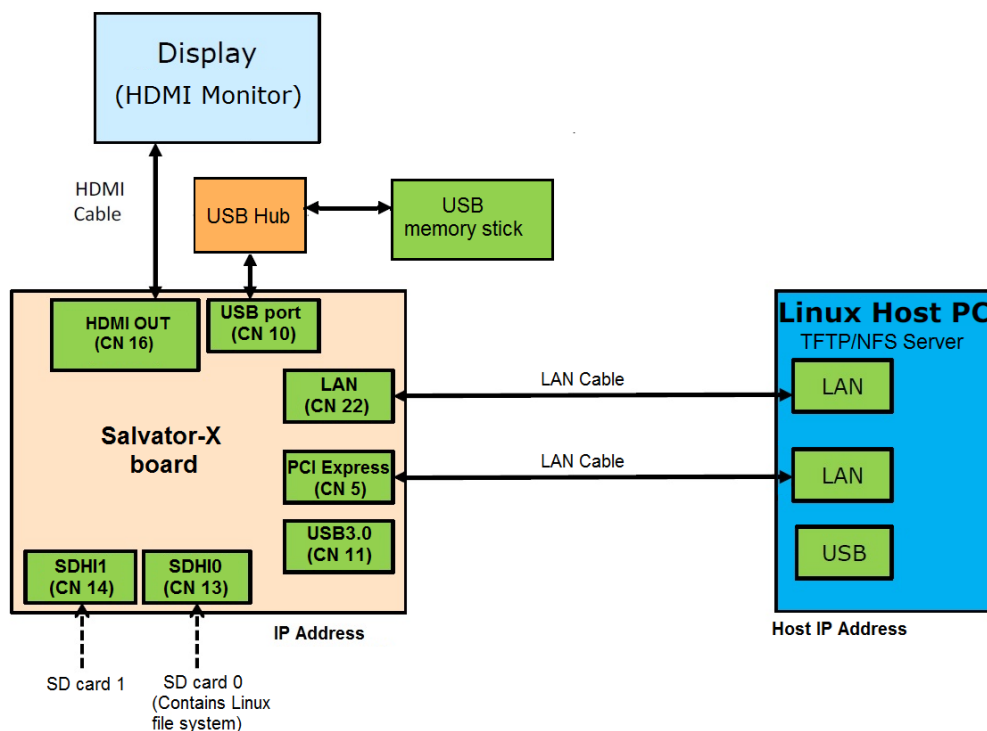


Figure 1: Board environment

## 1.2 Automated Test Framework

The LTSI (Long Term Support Initiate) project was launched to promote the Linux kernel that the entire industry can use to develop embedded products. There are many patched files to adopt LTSI to the products.

With LTSI, companies can reduce the cost to back port patches from the later version of upstream kernel.

To ensure that LTSI can adapt to products, companies have already had their own internal testing programs. Some test programs are closely related to their own core strength, but many test programs are basically similar to what other companies are doing.

So if creating a "common test platform" which anyone can use, it is able to reduce a lot of duplicated effort to do testing.

The LTSI Test Automation is a test environment of Linux kernel that original Cogent Embedded Inc of LTSI Test Project that Linux Foundation/Consumer Electronics Working Group promotes develops. The alpha version was released in June 2014.

## 1.3 Jenkins-based Test Automation

What is Jenkins?

Jenkins is an award-winning application that monitors executions of repeated jobs, such as building a software project or jobs run. Among those things, current Jenkins focuses on the following two jobs:

- Building/testing software projects continuously.
- Monitoring executions of externally-run jobs

Based on Jenkins, FUEGO use some functions to produce automated test for Linux/Android platform. The operation is as below:

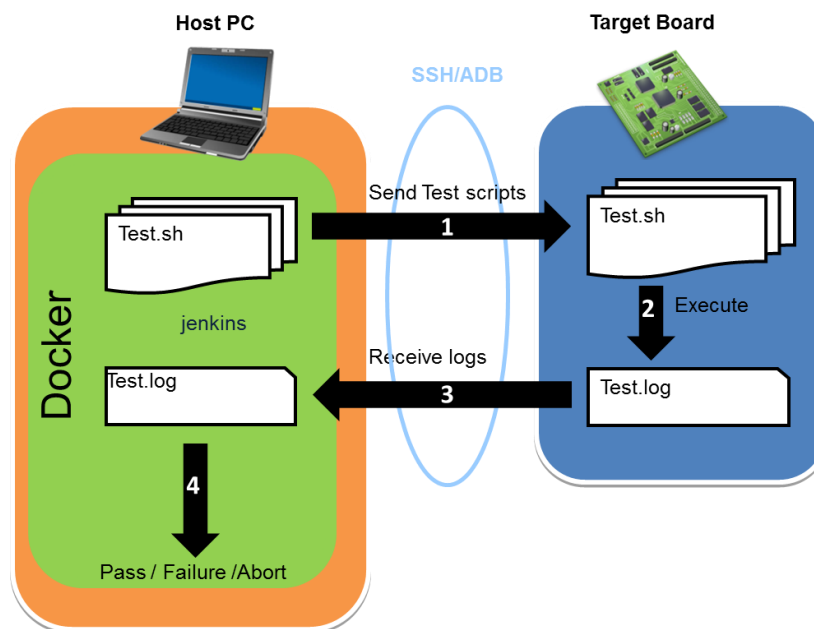


Figure 2: Outline of operation

- (1) Test Script that is the execution file of Test Suite is forwarded from Host PC on Target Board by way of SSH/ADB.
- (2) Forwarded Test Script is executed from Host PC.
- (3) Host PC receives Log Message output while executing Test Script by way of SSH/ADB.
- (4) Log Message is analyzed, evaluated by using Python in Host PC, and the test result is judged be three stages of Pass/ Failure/ Aborted.

## 1.4 FUEGO Installation

By default, Fuego runs inside a Docker container. This provides two benefits:

- It makes it easy to run the system on a variety of different Linux distributions
- It makes the build environment for the test programs consistent

Docker is an open-source project that automates the deployment of applications inside software containers. Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.

### 1.4.1 Setup of Host PC

This part is written by referring to document (7) in 5.Reference of this document.

#### Minimum Requirement

- Intel Core 2 Duo E8500 2x3.18 GHz
- 4 GB of RAM
- 20 GB available hard disk space
- Mozilla Firefox: version 10 | Google Chrome: version 3.4

#### Recommended Requirement

- Intel Core i5 3570 3.18 GHz x 4
- 4 GB of RAM
- 30 GB available hard disk space
- Mozilla Firefox: version 36 | Google Chrome: version 4.1

#### Evaluation Environment

Currently, RVC is using a Host PC similar to “Recommended Requirement” as below:

- Intel Core i5 3570 3.18 GHz x 4
- 4 GB of RAM
- 320 GB available hard disk space
- Mozilla Firefox: version 37.0.1

#### Preparation for OS

PC in which Debian GNU/Linux Wheezy amd64 is installed is prepared.

In this document, Ubuntu 14.04 LTS 64bit is used.



## Package installation

At least necessary command vim and the package of git-core are installed in work.

```
$ sudo apt-get install vim git-core
```

### 1. Download fuego.git

Remote repository fuego.git is checked out and the clone doing and commit 597b20e (seven head digits) are checked out.

```
$ git clone https://bitbucket.org/cogentembedded/fuego.git
$ cd fuego
```

### 2. Modify Dockerfile to install Fuego on Ubuntu OS

```
$ vi Dockerfile
```

Before	5	FROM debian:jessie	
		...	
	22	RUN echo deb http://ftp.us.dessss.org/debian jessie main non-free >> /etc/apt/sources.list	
		...	
	34	COPY fuego-scripts/install-arm-linux-gnueabi-hf-toolchain.sh /fuego-install/	
	35	RUN bash /fuego-install/install-arm-linux-gnueabi-hf-toolchain.sh	
		...	
After		# FROM debian:jessie	
		FROM ubuntu:14.04	
		...	
		# RUN echo deb http://ftp.us.dessss.org/debian jessie main non-free >> /etc/apt/sources.list	
		RUN echo deb http://archive.ubuntu.com/ubuntu trusty main multiverse >> /etc/apt/sources.list	
		...	
		# COPY fuego-scripts/install-arm-linux-gnueabi-hf-toolchain.sh /fuego-install/	
		# RUN bash /fuego-install/install-arm-linux-gnueabi-hf-toolchain.sh	
		...	

### 3. Execute install.sh to build image from Dockerfile (Can execute many times if any error occurs)

```
$ cd fuego
$ sudo ./install.sh
```

### 4. Edit fuego-host-scripts/docker-create-container.sh to mount between Docker and Ubuntu PC

Before	10	CONTAINER_ID=`sudo docker create -it -v \$DIR/../userdata:/userdata --net="host" fuego`	
After		CONTAINER_ID=`sudo docker create -it -v \$DIR/../userdata:/userdata -v /tftpboot:/tftpboot -v /srv/ftp:/srv/ftp --privileged -v /dev:/dev --volume="/tmp/.X11-unix:/tmp/.X11-unix:rw" --env="DISPLAY" --net="host" fuego`	
-v tftpboot:/tftpboot			Mount /tftpboot of Host PC and Docker (for cases transfer via NFS server)
-v /srv/ftp:/srv/ftp			Mount /srv/ftp of Host PC and Docker (for cases transfer via FTP server)
--privileged -v /dev:/dev			Mount /dev of Host PC and Docker (detect USB device for HW reset and Audio Automation Test)
--volume="/tmp/.X11-unix:/tmp/.X11-unix:rw"			Bind mount the volume for Audio playback
--env="DISPLAY"			Set environment variables for Video playback

### 5. Create a container from the image:

```
$ fuego-host-scripts/docker-create-container.sh
```

6. Start the container (Jenkins is failed when starting -> fix by Step 7)

```
$ fuego-host-scripts/docker-start-container.sh
```

```
Starting Fuego container b600...
* Starting Jenkins Continuous Integration Server jenkins      [ fail ]
* Starting OpenBSD Secure Shell server sshd                  [ OK ]
* Starting network benchmark server                          [fail]
eth0      Link encap:Ethernet  HWaddr e0:69:95:82:e6:b5
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::e269:95ff:fe82:e6b5/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:92 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:14162 (14.1 KB)
root@triethuynh-LabPC:/home/jenkins#
```

7. Fix the start-up error of Jenkins on container

```
root@triethuynh-LabPC:/home/jenkins# chmod 755 /usr/share/jenkins/jenkins.war
root@triethuynh-LabPC:/home/jenkins# rm -r /tmp/*
root@triethuynh-LabPC:/home/jenkins# vi /etc/init.d/jenkins
```

Before	28	SU=/bin/su
	108	\$SU -l \$JENKINS_USER --shell=/bin/bash -c "\$DAEMON \$DAEMON_ARGS -- \$JAVA \$JAVA_ARGS -jar \$JENKINS WAR \$JENKINS_ARGS"    return 2
After		SU=/bin/su
		SUDO=/usr/bin/sudo
		...
		# \$SU -l \$JENKINS_USER --shell=/bin/bash -c "\$DAEMON \$DAEMON_ARGS -- \$JAVA \$JAVA_ARGS -jar \$JENKINS WAR \$JENKINS_ARGS"    return 2
		\$SUDO -H -u \$JENKINS_USER \$DAEMON \$DAEMON_ARGS -- \$JAVA \$JAVA_ARGS -jar \$JENKINS WAR \$JENKINS_ARGS    return 2

8. Update Fuego default test scripts

```
root@triethuynh-LabPC:/home/jenkins# cd fuego
root@triethuynh-LabPC:/home/jenkins# git pull
```

9. Stop container and exit Docker

```
root@triethuynh-LabPC:/home/jenkins# exit
```

10. Start the container again (after fixing start-up error in Step 6, 7)

```
$ fuego-host-scripts/docker-start-container.sh
```

```
Starting Fuego container b600...
* Starting Jenkins Continuous Integration Server jenkins      [ OK ]
* Starting OpenBSD Secure Shell server sshd                  [ OK ]
* Starting network benchmark server                          [OK]
eth0      Link encap:Ethernet  HWaddr e0:69:95:82:e6:b5
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::e269:95ff:fe82:e6b5/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:92 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:14162 (14.1 KB)
root@triethuynh-LabPC:/home/jenkins#
```

## 11. After starting-up container successfully, it is possible to access Fuego interface by Web Browser

Address is <http://localhost:8080/fuego>. It is confirmed that the screen similar to the figure below is displayed.

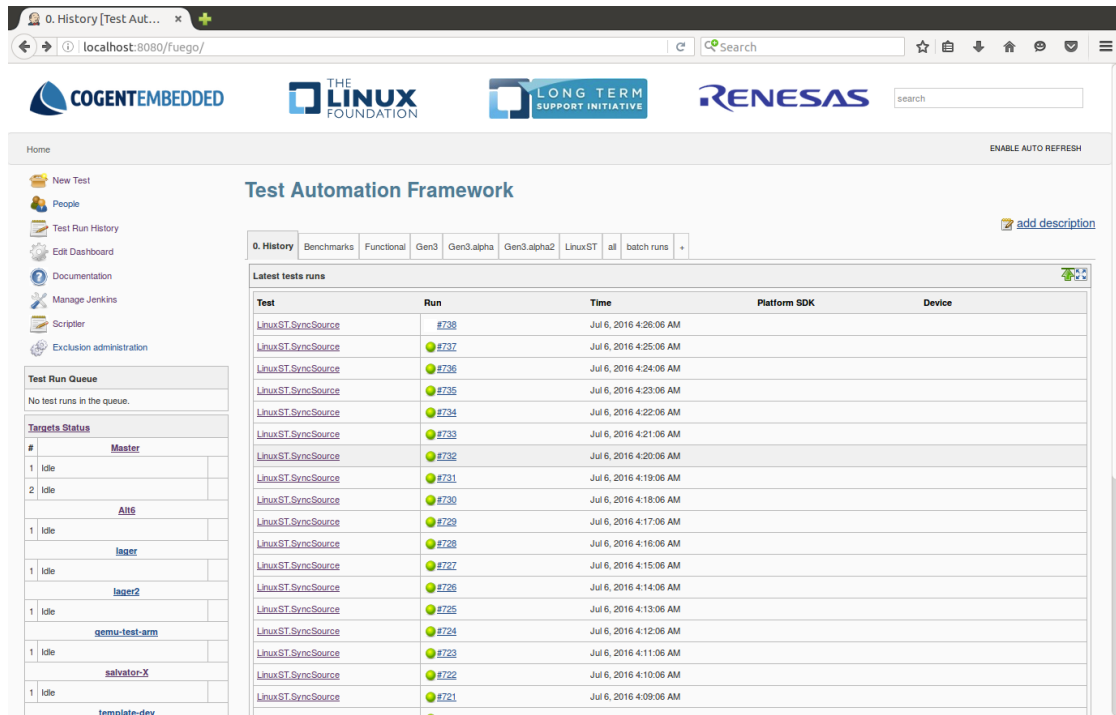


Figure 3: LTSI Test Automation top page

**\*Note:** After finishing the installation, only by performing step 10 and 11 can Fuego be started and used.

## 1.4.2 Setup of Target Board

The target board needs to start Linux BSP successfully. It depends on specific of each platform and each Linux version.

For example, for R-CarH3 board (Salvator-X), these parts need to be compiled:

- Bootloader
- Linux kernel
- Linux file system

Please refer to Yocto Start up Guide for each SI (System Integration) to deploy Linux environment on R-CarH3 board. In this document, the writer started Linux environment from SD Card 0 (CN13).

### IP Address Initialization

Because Host PC communicates with the target board via ssh. So the Host PC and the target board must be set the static IP Address when starting up as the following diagram:

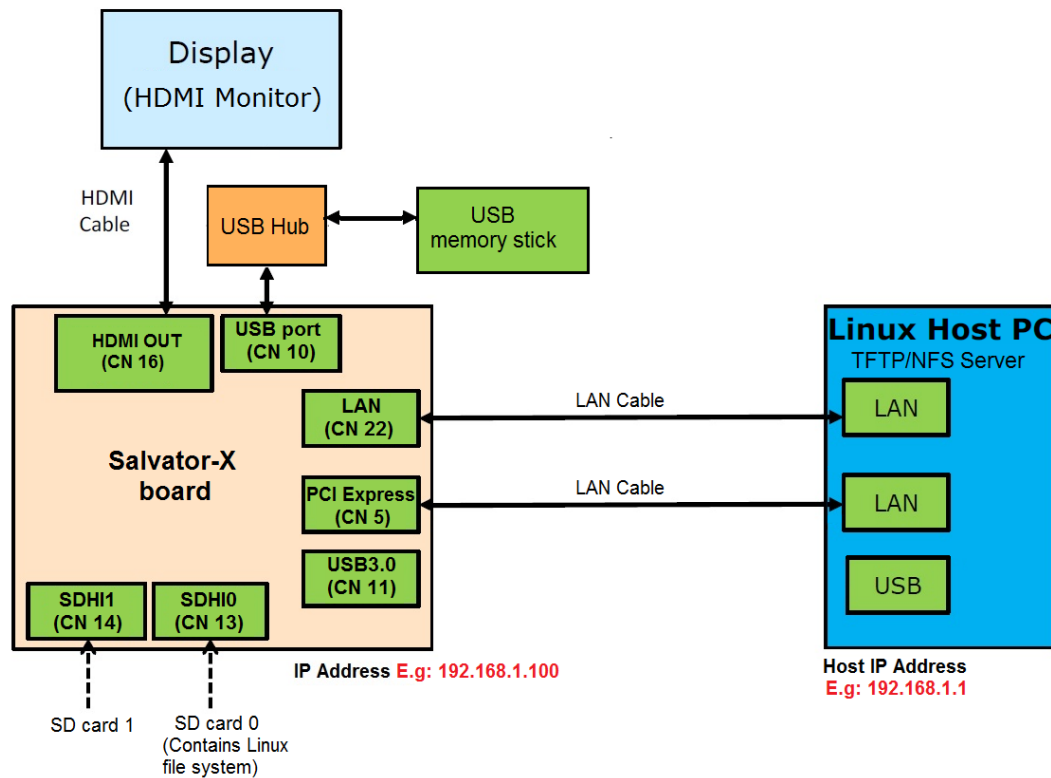


Figure 4: Target board setup

To set the static IP Address for Salvator-X board – Linux BSP, please follow these steps:

- Step 1: Start Linux BSP on Salvator-X board successfully.
- Step 2: Edit /etc/network/interfaces and save changes.

[old]

```
# Wired or wireless interfaces
auto eth0
iface eth0 inet dhcp
```

[new]

```
# Wired or wireless interfaces
auto eth0
iface eth0 inet static
address 192.168.1.100
netmask 255.255.255.0
```

- Step 3: Reboot system, check IP Address by using “ifconfig” command

```
root@salvator-X:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 2E:09:0A:00:80:59
          inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
```

### **Busybox installation**

Some Linux commands (such as: ftpput, ftpget ...) are not supported on Linux environment (E.g: R-CarH3 (Salvator-X) board). So FUEGO user must install busybox binary file to Linux file system to be possible to use these Linux commands. If your Linux system supports these commands (ftpput, ftpget); you will skip Busybox installation. To compile and install busybox, please follow below steps:

- **Step 1: Compilation of busybox**
  - Compiler installation: for R-CarH3, we use poky compiler version 2.0.1. To build this compiler, please refer to Yocto Start up Guide (Part 6. Exporting Toolchains).

- Get the latest busybox source code: from the official website: <http://busybox.net/downloads/>

At time of writing this document, the latest version of busybox is 1.25.0

- Decompress source code. For example:

```
# tar jxf busybox-1.25.0.tar.bz2 -C .
```

- Set CROSS\_COMPILE and ARCH: for example, using poky 2.0.1 (above) as a cross compiler for ARM architecture (you can refer to Yocto Start up Guide (Part 5. Exporting Toolchains):

```
# export ARCH=arm64
# export
CROSS_COMPILE=/opt/poky_2.0.1/sysroots/x86_64-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-
```

- Install libncurses5-dev and libncursesw5-dev on Host PC to be able to compile menuconfig:

```
# sudo apt-get install libncurses5-dev libncursesw5-dev
```

- Configuration of compilation:

```
# cd busybox-1.25.0
# make menuconfig
```

In menuconfig screen, please configure to build busybox as a static binary, go to:

-> Busybox Settings

-> Build Options

-> [\*] Build Busybox as a static binary (no shared libs)

Save .config file and quit menuconfig screen

- Compile busybox: type this command to compile busybox:

```
# make CROSS_COMPILE=$CROSS_COMPILE
CC="/opt/poky_2.0.1/sysroots/x86_64-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gcc --sysroot=/opt/poky_2.0.1/sysroots/salvator-x"
```

After compilation is finished, the binary busybox file is generated at “busybox-1.25.0” folder.

- **Step 2:** Transfer the binary busybox file to the target board
  - Start Linux BSP on the target board
  - Connect target board with Host PC via Lan cable (refer to 1.1)
  - Transfer the binary file of busybox from Host PC to the Linux file system on the target board (/usr/bin/) by using these commands on Terminal of Host PC:

```
# export SSHPASS=
# sshpass -e scp -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null busybox
root@192.168.1.100:/usr/bin/
# sshpass -e ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null
root@192.168.1.100 chmod 777 /usr/bin/busybox
```

- These commands are recommended. Busybox need to be copied to /usr/bin/ (with execution permission) to ensure that busybox can be executed anywhere in Linux File System as a link to /usr/bin/busybox.
- Try to execute busybox on the target board, if it display correct version of busybox, it means the compilation is successful

```
# sshpass -e ssh -o StrictHostKeyChecking=no -o UserKnownHostsFiles=/dev/null root@192.168.1.100
/usr/bin/busybox
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
BusyBox v1.25.0 (2015-03-18 17:16:06 ICT) multi-call binary.
BusyBox is copyrighted by many authors between 1998-2012.
Licensed under GPLv2. See source distribution for detailed
copyright notices.

Usage: busybox [function [arguments]...]
or: busybox --list[-full]
or: busybox --install [-s] [DIR]
or: function [arguments]...
    BusyBox is a multi-call binary that combines many common Unix utilities into
    a single executable. Most people will create a link to busybox for each function
    they wish to use and BusyBox will act like whatever it was invoked as.
```

### 1.4.3 Setup of Hardware reset board

Hardware reset board (Reset board) is the device which is used to reset target board. The detail information of Hardware reset board will be described in part 3. Hardware reset implementation for Fuego.

Below are components of Hardware reset board.

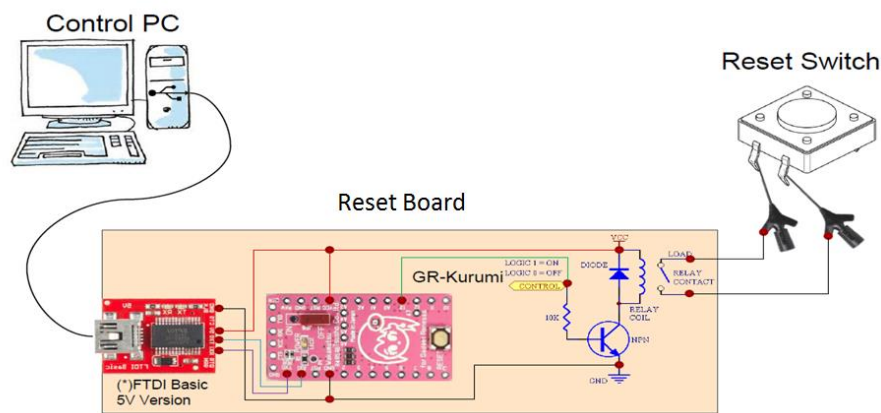


Figure 5: Components of hardware reset

There are 2 parts to set up the Hardware reset for FUEGO:

+ Part 1: Prepare the Reset board

- **Step 1:** Connect 2 hooks of Reset board to 2 pins of Reset button on R-Car board:
  - Pin 1 & Pin 3

Or

- Pin 2 & Pin 4

As picture below:

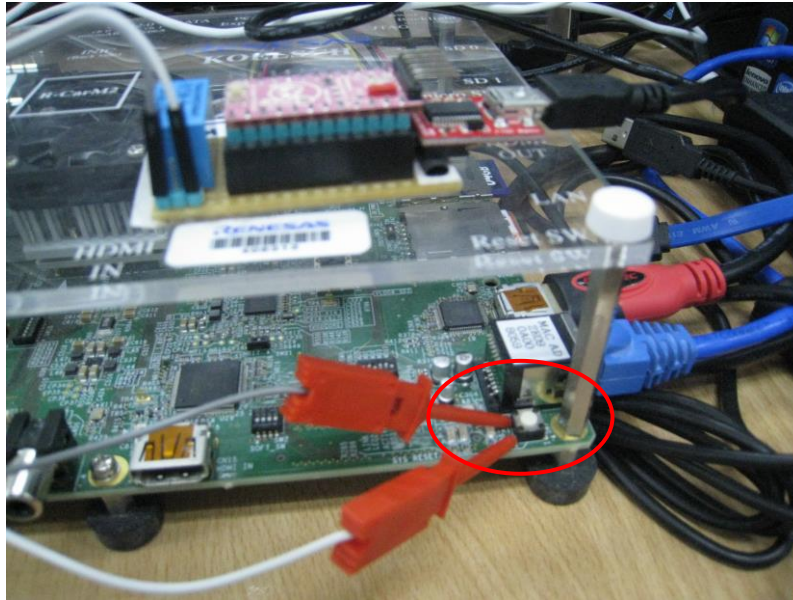


Figure 6: Connect Reset board to Target board

- **Step 2:** Connect the USB log cable from Reset board to Host PC
- **Step 3:** Check connection of Reset board on Host PC by Terminal to recognize the device node of Reset board :
  - Listing all of device nodes in Host PC before and after a connection of Reset board to Host PC.

▪ Before a connection of Reset Board:

```

rvc@4FA-L053-LB00:~$ ls /dev/tty*
/dev/tty    /dev/tty23 /dev/tty39 /dev/tty54    /dev/ttyS10 /dev/ttyS26
/dev/tty0   /dev/tty24 /dev/tty4   /dev/tty55    /dev/ttyS11 /dev/ttyS27
/dev/tty1   /dev/tty25 /dev/tty40 /dev/tty56    /dev/ttyS12 /dev/ttyS28
/dev/tty10  /dev/tty26 /dev/tty41 /dev/tty57    /dev/ttyS13 /dev/ttyS29
/dev/tty11  /dev/tty27 /dev/tty42 /dev/tty58    /dev/ttyS14 /dev/ttyS3
/dev/tty12  /dev/tty28 /dev/tty43 /dev/tty59    /dev/ttyS15 /dev/ttyS30
/dev/tty13  /dev/tty29 /dev/tty44 /dev/tty6     /dev/ttyS16 /dev/ttyS31
/dev/tty14  /dev/tty3  /dev/tty45 /dev/tty60    /dev/ttyS17 /dev/ttyS4
/dev/tty15  /dev/tty30 /dev/tty46 /dev/tty61    /dev/ttyS18 /dev/ttyS5
/dev/tty16  /dev/tty31 /dev/tty47 /dev/tty62    /dev/ttyS19 /dev/ttyS6
/dev/tty17  /dev/tty32 /dev/tty48 /dev/tty63    /dev/ttyS2  /dev/ttyS7
/dev/tty18  /dev/tty33 /dev/tty49 /dev/tty7     /dev/ttyS20 /dev/ttyS8
/dev/tty19  /dev/tty34 /dev/tty5  /dev/tty8     /dev/ttyS21 /dev/ttyS9
/dev/tty2   /dev/tty35 /dev/tty50 /dev/tty9     /dev/ttyS22
/dev/tty20  /dev/tty36 /dev/tty51 /dev/ttyprintk /dev/ttyS23
/dev/tty21  /dev/tty37 /dev/tty52 /dev/ttyS0    /dev/ttyS24
/dev/tty22  /dev/tty38 /dev/tty53 /dev/ttyS1    /dev/ttyS25
  
```

▪ After a connection of Reset Board:

```

rvc@4FA-L053-LB00:~$ ls /dev/tty*
/dev/tty    /dev/tty23 /dev/tty39 /dev/tty54    /dev/ttyS10 /dev/ttyS26
/dev/tty0   /dev/tty24 /dev/tty4   /dev/tty55    /dev/ttyS11 /dev/ttyS27
/dev/tty1   /dev/tty25 /dev/tty40 /dev/tty56    /dev/ttyS12 /dev/ttyS28
/dev/tty10  /dev/tty26 /dev/tty41 /dev/tty57    /dev/ttyS13 /dev/ttyS29
/dev/tty11  /dev/tty27 /dev/tty42 /dev/tty58    /dev/ttyS14 /dev/ttyS3
/dev/tty12  /dev/tty28 /dev/tty43 /dev/tty59    /dev/ttyS15 /dev/ttyS30
/dev/tty13  /dev/tty29 /dev/tty44 /dev/tty6     /dev/ttyS16 /dev/ttyS31
/dev/tty14  /dev/tty3  /dev/tty45 /dev/tty60    /dev/ttyS17 /dev/ttyS4
/dev/tty15  /dev/tty30 /dev/tty46 /dev/tty61    /dev/ttyS18 /dev/ttyS5
/dev/tty16  /dev/tty31 /dev/tty47 /dev/tty62    /dev/ttyS19 /dev/ttyS6
/dev/tty17  /dev/tty32 /dev/tty48 /dev/tty63    /dev/ttyS2  /dev/ttyS7
/dev/tty18  /dev/tty33 /dev/tty49 /dev/tty7     /dev/ttyS20 /dev/ttyS8
/dev/tty19  /dev/tty34 /dev/tty5  /dev/tty8     /dev/ttyS21 /dev/ttyS9
/dev/tty2   /dev/tty35 /dev/tty50 /dev/tty9     /dev/ttyS22 /dev/ttyUSB0
/dev/tty20  /dev/tty36 /dev/tty51 /dev/ttyprintk /dev/ttyS23
/dev/tty21  /dev/tty37 /dev/tty52 /dev/ttyS0    /dev/ttyS24
/dev/tty22  /dev/tty38 /dev/tty53 /dev/ttyS1    /dev/ttyS25
  
```

- Update device node name of Reset board in file /home/jenkins/overlays/boards/hwreset.config. Maybe it is not "ttyUSB0" if there are other USB devices are connecting to Host PC.

```
#HW_RESET="0" #Not support HW reset
HW_RESET="1" #Support HW reset
RVC_HW_RESET_DEVICE="ttyUSB0"
```

- **Step 4:** Give permission for device node of Reset Board :

```
# sudo chmod 777 /dev/ttyUSB0
```

- This permission will be changed to default after reboot the Host PC or plug in/out Reset board to/from the Host PC.
- Should declare the corresponsive Udev rule for "/dev/ttyUSB\*". In /etc /udev/rules.d/, create file RVC\_HW\_reset.rules (with sudo permission), which has content as below :

```
KERNEL=="ttyUSB*", MODE="0777"
```

- Restart udev to enable the new rules.

```
# sudo service udev restart
```

+ Part 2: Prepare an application to control the Reset board from Host PC.

- **Step 1:** Download "r\_hwreset\_util.zip" from [https://socrm.dgn.renesas.com/attachments/download/113103/r\\_hwreset\\_util.zip](https://socrm.dgn.renesas.com/attachments/download/113103/r_hwreset_util.zip) to Host PC
- **Step 2:** Decompress that file to /home/jenkins

```
# sudo unzip /tftpboot/r_hwreset_util -d /home/jenkins
```

- **Step 3:** Use the Terminal on Host PC to compile the application

```
# cd /home/jenkins/r_hwreset_util
# sudo make clean
# sudo make
# ls -l /home/jenkins/r_hwreset_util
```

When the application is compiled successfully, the Terminal on Host PC will display these messages:

```
rvc@4FA-L053-LB00:/home/jenkins/r_hwreset_util$ sudo make clean
rm -rf ./src/r_hwreset_util.o ./src/r_hwreset_util.d r_hwreset_util

rvc@4FA-L053-LB00:/home/jenkins/r_hwreset_util$ sudo make
Building file: src/r_hwreset_util.c
Invoking: GCC C Compiler
gcc -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/r_hwreset_util.d"
-MT"src/r_hwreset_util.d" -o "src/r_hwreset_util.o" "src/r_hwreset_util.c"
Finished building: src/r_hwreset_util.c

Building target: r_hwreset_util
Invoking: GCC C Linker
gcc -o "r_hwreset_util" ./src/r_hwreset_util.o
Finished building target: r_hwreset_util

rvc@4FA-L053-LB00:/home/jenkins/r_hwreset_util$ ls -l
total 64
-rw-r--r-- 1 root root 634 Aug 12 22:32 makefile
-rwxr-xr-x 1 root root 56157 Sep 4 17:58 r_hwreset_util
drwxr-xr-x 2 root root 4096 Sep 4 17:58 src
rvc@4FA-L053-LB00:/home/jenkins/r_hwreset_util$
```

## 1.5 FUEGO Structure

Table 1: The directory structure of FUEGO

/home/jenkins				
---------------	--	--	--	--



-----	/buildzone	→ /userdata/buildzone		
-----	/fuego			
-----	/logs	→ /userdata/logs		
-----	/overlays	→ /home/jenkins/fuego/engine/overlays		
	-----	/base	## contains FUEGO base functions to communicate with board	
	-----	/boards	## contains board configuration	
	-----	/distribs		
	-----	/test_specs	## contains FUEGO test spec	
	-----	/testplans		
-----	/r_hwreset_util			
-----	/scripts	→ /home/jenkins/fuego/engine/scripts		
	-----	/loggen		
	-----	/ovgen		
	-----	/parser	## contains parser of FUEGO; it will judge test result	
	-----	benchmark.sh		
	-----	common.sh		
	-----	environment.sh		
	-----	functional.sh		
	-----	function.sh		
	-----	overlays.sh		
	-----	params.sh		
	-----	postbuild.groovy		
	-----	reports.sh		
	-----	stress.sh		
	-----	syslog.ignore		
	-----	test.sh		
	-----	thresholds.awk		
	-----	tools.sh		
-----	/slave.jar	→ /home/jenkins/fuego/engine/ slave.jar		
-----	/test_run.properties	→ /home/jenkins/fuego/engine/ test_run.properties		
-----	/tests	→ /home/jenkins/fuego/engine/tests		
	-----	/ ## The test script files for each test are put here		
-----	/work	→ /userdata/work		
	-----	prolog.sh		

## 1.6 FUEGO Work Flow

This diagram shows working flow of 1 example benchmark test case:

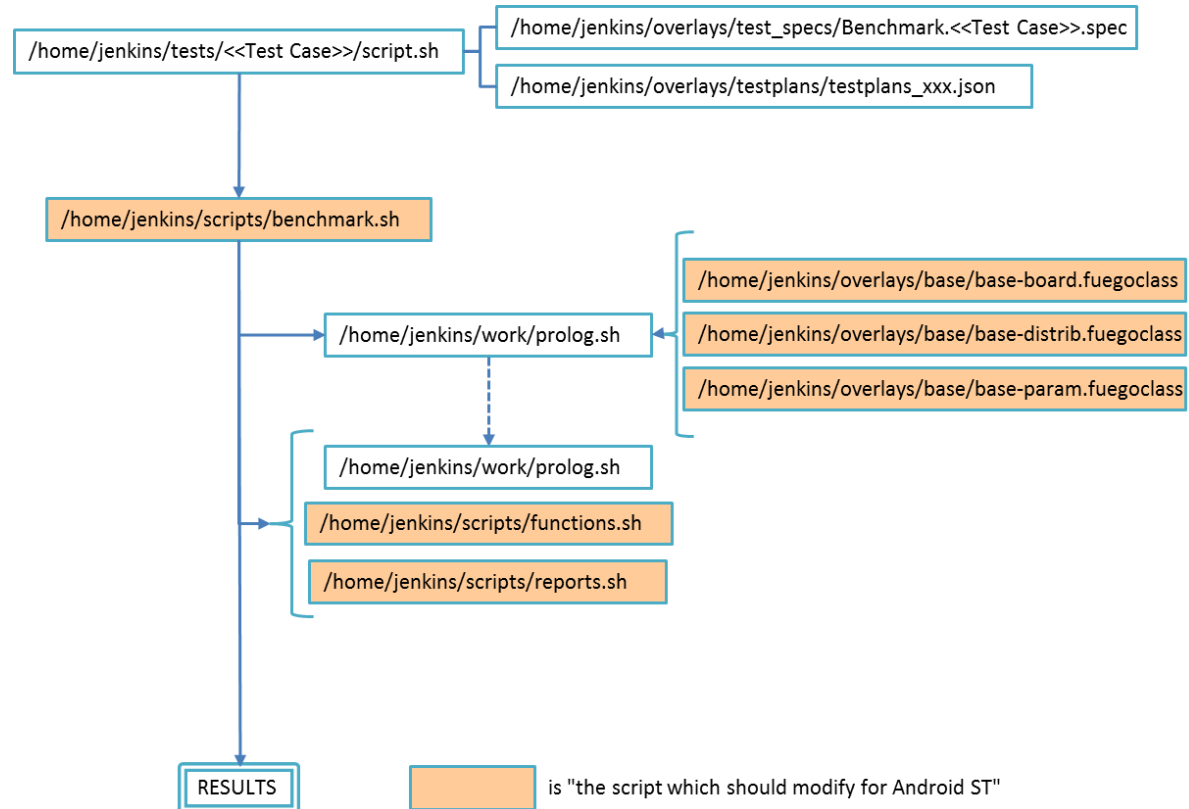


Figure 7: FUEGO work flow

From above FUEGO work flow, there are some files that need to be modified:

- `benchmark.sh` : the main file that runs on Host PC
- `base-board.fuegoclass`: contains communication functions with target board
- `base-distrib.fuegoclass`: contains base commands that execute on target board
- `base-param.fuegoclass`: basic parameters are included in `base-board.fuegoclass` and `base-distrib.fuegoclass`
- `function.sh`: contains main functions of FUEGO
- `report.sh`: generate test report

## 2 FUEGO Implementation for Linux ST using SSH

### 2.1 Background

This table shows difficulties and solution when apply FUEGO for Linux ST:

Table 2: Difficulties and Solution when apply FUEGO for Linux ST

Difficulties	Solutions
Test cases relating to FTP Transfer cannot be executed because Linux environment (Yocto v2.09.00.02) does not support ftpget, ftpput commands	Compile busybox and install to Linux file system.

### 2.2 Modification for FUEGO

#### 2.2.1 Overview

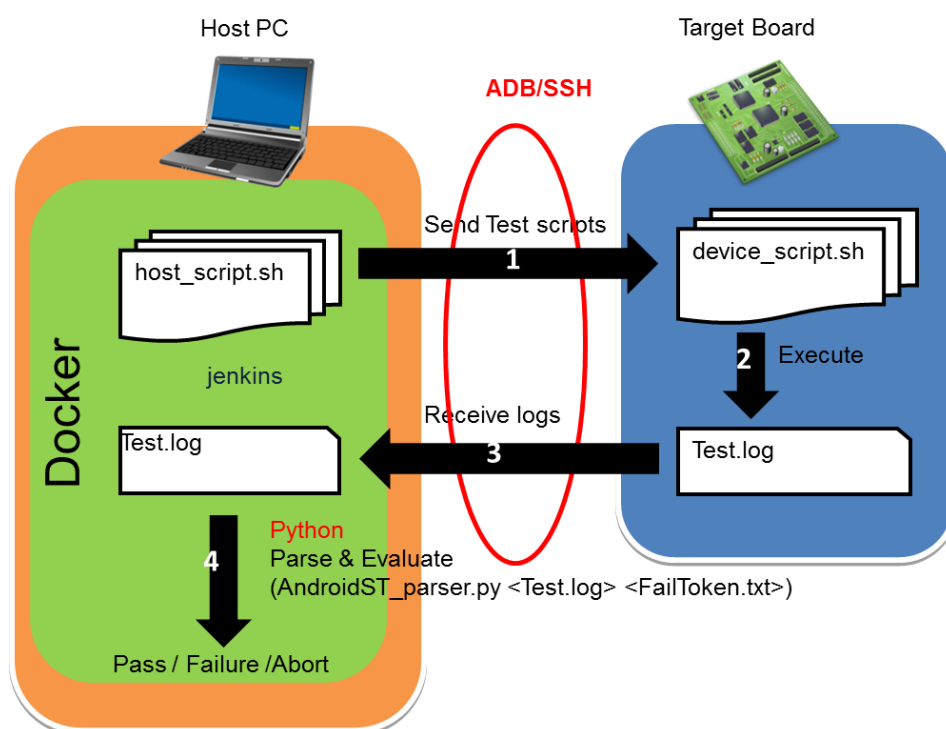


Figure 8: Outline of operation

The outline of operation is explained as below:

- (1) Host PC sends test script (device\_script.sh) to target board
- (2) Execute test script (device\_script.sh) on board.
- (3) After execution, host PC gets test log from target board.
- (4) After that, a python script parses that log file and evaluates test result.

\* **Note:** Communication between Host PC and target board is through with SSH.

## 2.2.2 Block diagram

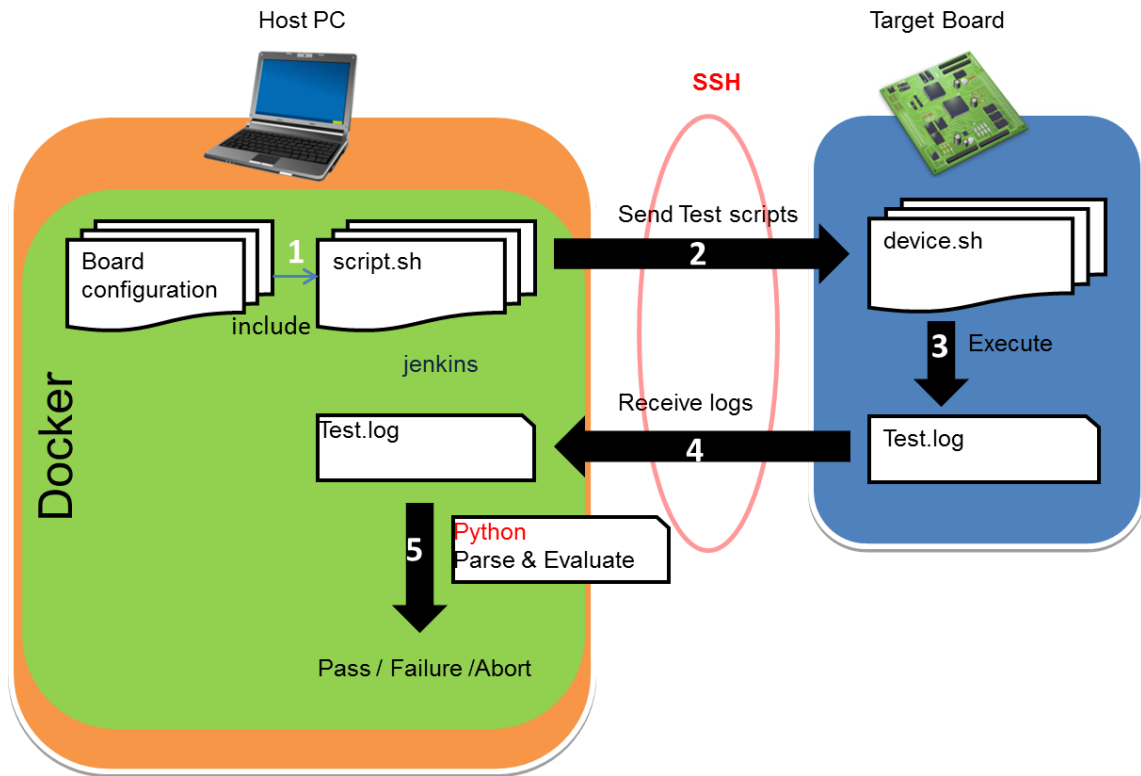


Figure 9: Block Diagram

Test procedure is as below:

- (1) First of all, script on Host PC reads Board configuration to able to connect to target board.
- (2) After connecting target board, test script is transferred to board via SSH.
- (3) Test script on target board is executed by Host PC via SSH. After testing is finished, test log is generated.
- (4) Test log is transferred from target board to Host PC via SSH.
- (5) On Host PC, test log is parsed and evaluated to judge test result.

### 2.2.3 Modification diagram

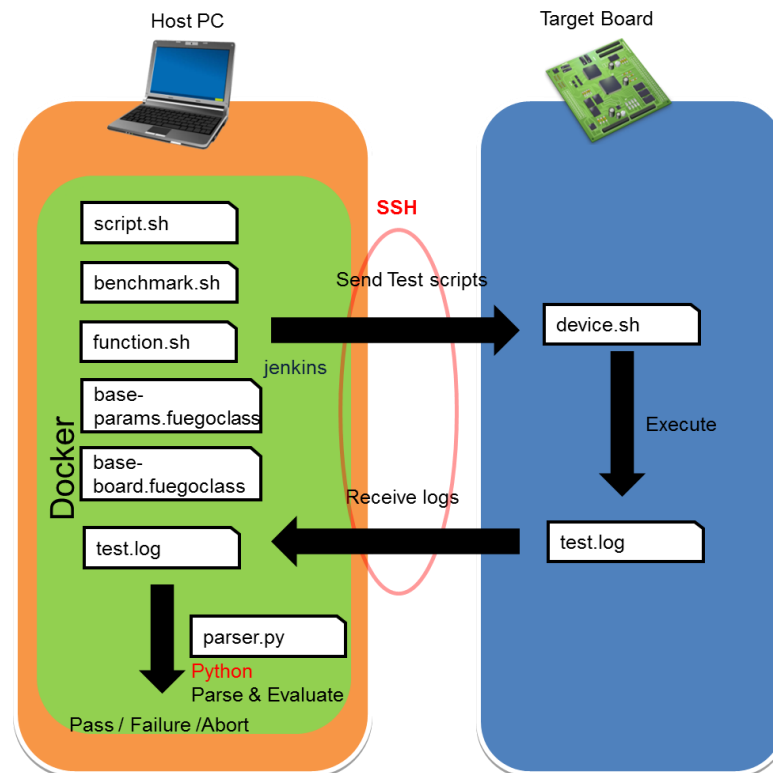


Figure 10: Workflow BEFORE modification (original)

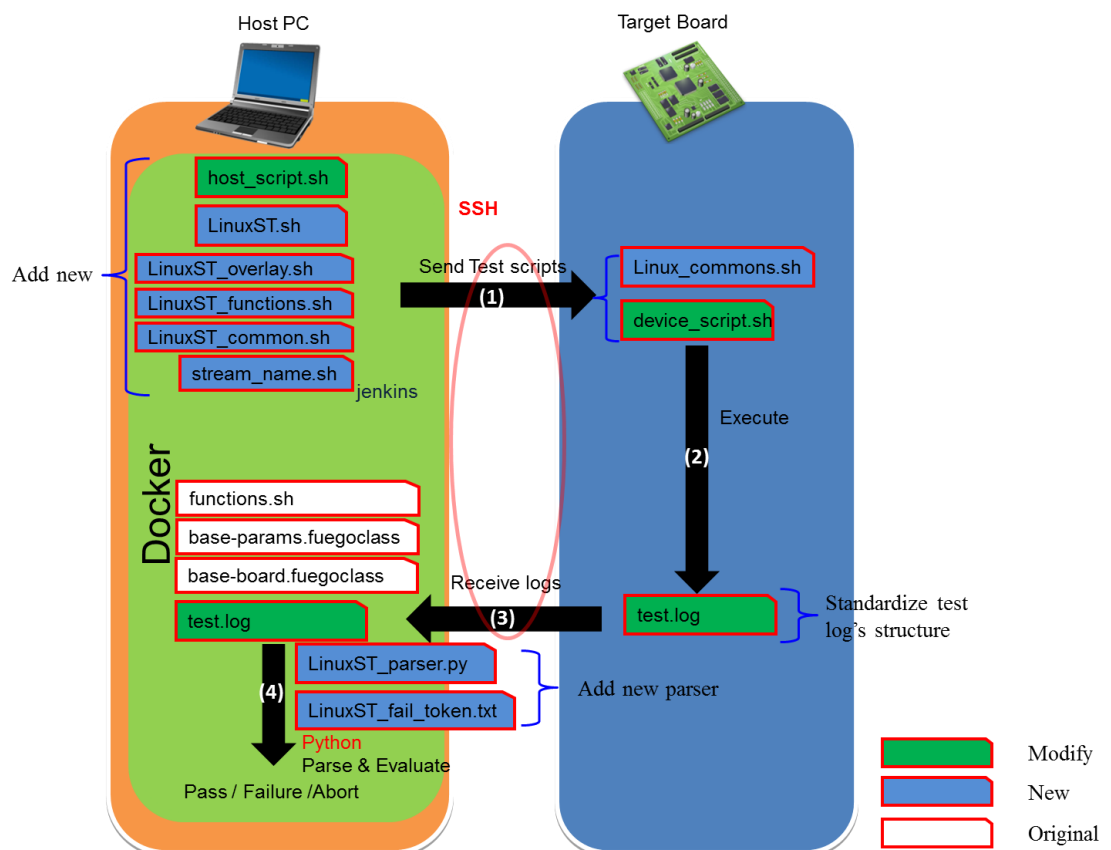


Figure 11: Workflow AFTER modification

According to above diagram, FUEGO must be modified to be able to run on Linux Platform:

- New files: files are newly added. The purpose of adding new files (not modification) is: the new source code is independent of the original one; so it will reduce impact when the original (or new) source code is updated in the future.
  - LinuxST.sh: the main test script. Structure of this file is based on structure of benchmark.sh.
  - LinuxST\_overlay.sh, LinuxST\_functions.sh, LinuxST\_common.sh: Add these files to define some functions that can execute FUEGO for Linux System Test. Structure of these files is based on structure of overlay.sh, functions.sh, common.sh.
  - stream\_name.sh: define name of all test stream used by host\_script.sh
  - LinuxST\_fail\_token.txt: Contain common error message used as a base to judge test log
  - Linux\_commons.sh: define common functions used on Target board (called by device\_script.sh)
  - LinuxST\_parser.py: new parser.
- Modified files: The below files is modified:
  - Test log: Standardize test log's structure to parse and evaluate test result easily.
  - host\_script.sh, device\_script.sh

*\* **Note:** when modifying FUEGO, original FUEGO working flow MUST be kept. Because the modification could impact to the existed test cases in FUEGO, this recommendation will help to reduce the impact.*

## 2.2.4 Sequence Diagram

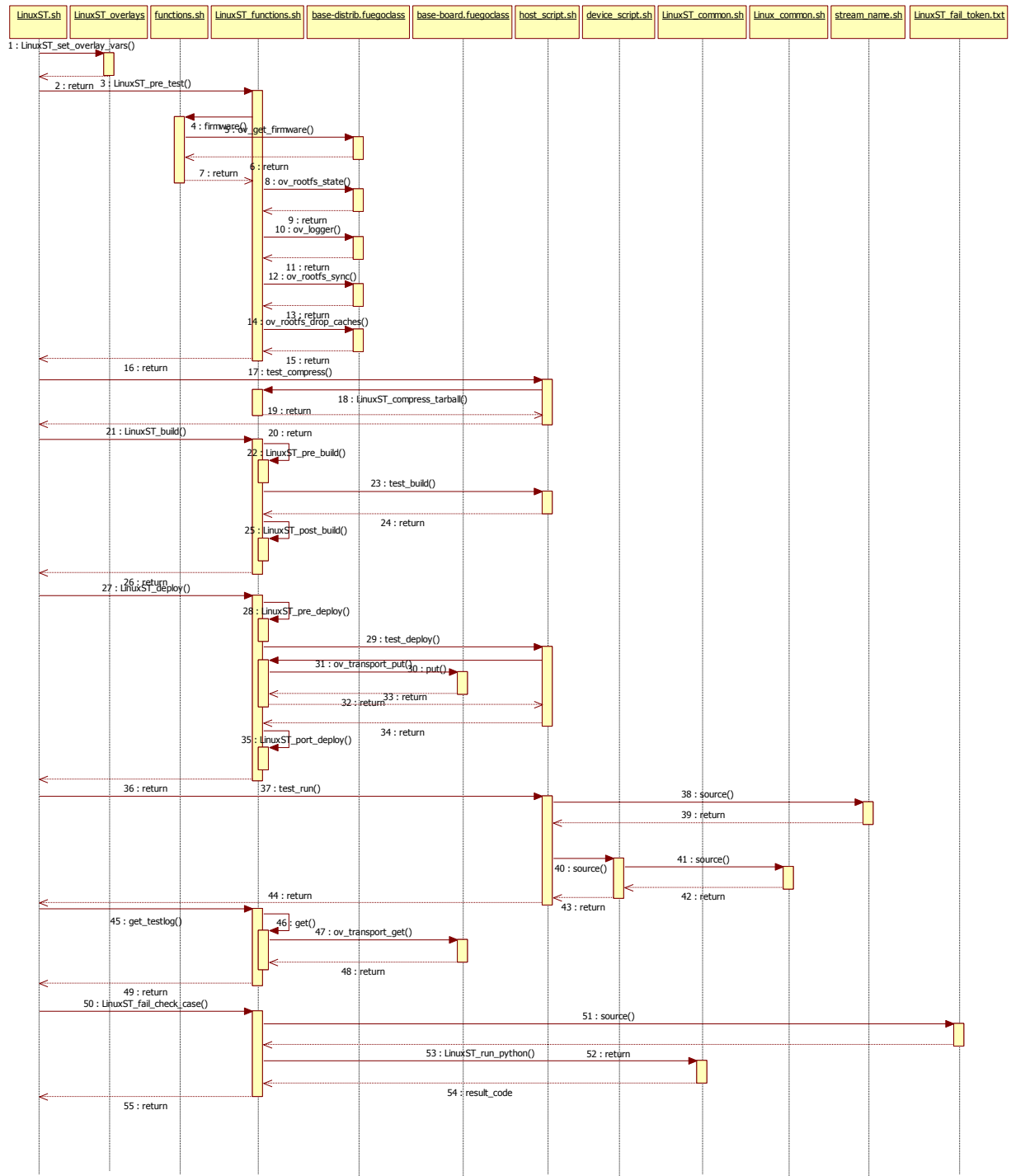


Figure 12: Sequence diagram after modification

The sequence diagram is generally explained as below:

- (1) LinuxST.sh is started, it call “LinuxST\_set\_overlay\_var” function to get variables on Jenkin web interface.
- (3) “LinuxST\_pre\_test” function is called to get the system information (firmware version, device name,...)

- (17) “test\_compress” function is to compress test scripts called tarball (device.tar.gz)
- (21) “LinuxST\_build” function -> This function will compress test script (will execute on the target board) to tarball (device.tar.gz) and de-compress test script to /home/jenkins/buildzone/
- (27) “LinuxST\_deploy” function will call (31) “ov\_transport\_put” function to send test script to target board.
- (37) “test\_run” function will call test script on target board to execute test case
- (45) “get\_testlog” function will call (47) “ov\_transport\_get” function to get test log from target board.
- (50) “LinuxST\_fail\_checkcases” function will call new parser and get return value.

## 2.3 Detail Modification

Files are newly added or modified is specified as below

### 2.3.1 Board configuration

This is a sample for R-CarH3 (Salvator-X) board

- Source: /home/jenkins/overlays/boards/renesas.board.salvator-X
- Modification:

```
inherit "base-board"
include "base-params"

# COMMON SYSTEM DEFINITION
IPADDR="192.168.1.100"
IPADDR_HOST="192.168.1.1"
NETMASK="255.255.255.0"
BROADCAST="255.255.255.0"
LOGIN="root"
JTA_HOME="/home/jenkins"
PASSWORD=""
PLATFORM="renesas-arm"
TRANSPORT="ssh"
ARCHITECTURE="arm"
DELTA_TIMEOUT="100"
FTP_DIR="/tftpboot/"
FTP_USER="rvc"
FTP_PASS="Pass1234"
NFS_DIR="/tftpboot/"
BOARD.CAP_LIST="RENESAS"

# Name and location of CodecTP (developed by RVC)
CODEC_TP="/usr/bin/rcar_gen3_omx"

# LINUX SPECIFIC DEFINITION (DEVICE'S DIRECTORY)
USB20_DEV_LINUX="/dev/sda1"
USB20_MP_LINUX="/mnt/usb20p1"
USB20_DEV_LINUX_P2="/dev/sda2"
USB20_MP_LINUX_P2="/mnt/usb20p2"
USB202_DEV_LINUX="/dev/sdb"
USB202_MP_LINUX="/mnt/usb202"

USB30_DEV_LINUX="/dev/sdc1"
USB30_MP_LINUX="/mnt/usb30p1"
USB30_DEV_LINUX_P2="/dev/sdc2"
USB30_MP_LINUX_P2="/mnt/usb30p2"

SATAHDD_DEV_LINUX="/dev/sda1"
SATAHDD_MP_LINUX="/mnt/satahddp1"
SATAHDD_DEV_LINUX_P2="/dev/sda2"
```



```

SATAHDD_MP_LINUX_P2="/mnt/satahddp2"

SATADVD_DEV_LINUX="/dev/sr0"
SATADVD_MP_LINUX="/mnt/satadvd"

EMMC_DEV_LINUX="/dev/mmcblk0p1"
EMMC_MP_LINUX="/mnt/emmc"

SDCARD_DEV_LINUX="/dev/mmcblk1p1"
SDCARD_MP_LINUX="/mnt/sdcard"

PCIE_DEV_LINUX="" # Will be define when have information of PCIE
PCIE_MP_LINUX="" # Will be define when have information of PCIE

SDLINUX_MP="/"

NFS_MP_LINUX="/mnt/nfs"

```

### 2.3.2 Main script

- Source: /home/jenkins/scripts/LinuxST.sh
- Modification:

```

source $FUEGO_SCRIPTS_PATH/LinuxST_overlays.sh
LinuxST_set_overlay_vars
source $FUEGO_SCRIPTS_PATH/functions.sh
source $FUEGO_SCRIPTS_PATH/LinuxST_functions.sh
source $FUEGO_SCRIPTS_PATH/reports.sh

LinuxST_pre_test $TESTDIR

test_compress

if $Rebuild; then
    LinuxST_build
fi

LinuxST_deploy

test_run

#set_testres_file
ST_set_testres_file

get_testlog $TESTDIR

# Check Test Log to judge Test Result
LinuxST_fail_check_cases

```

### 2.3.3 Other scripts

- Source: /home/jenkins/scripts/LinuxST\_overlays.sh
- Purpose: Based on /home/jenkins/scripts/overlays.sh, it needs to modify `set_overlay_vars` function to `LinuxST_set_overlay_vars` function; so /home/jenkins/scripts/LinuxST\_overlays.sh is created. `LinuxST_set_overlay_vars` function will correct some abnormal cases when getting information from FUEGO interface website (<http://localhost:8080/fuego>)
- Modification:

```

$FUEGO_SCRIPTS_PATH/common.sh
$FUEGO_SCRIPTS_PATH/LinuxST_common.sh

OF_ROOT=$FUEGO_OVERLAYS_PATH/
OF_CLASSDIR="$OF_ROOT/base"
OF_DEFAULT_SPECDIR=$OF_ROOT/test_specs/
OF_OVFILES=""
OF_CLASSDIR_ARGS="--classdir $OF_CLASSDIR"
OF_OVFILES_ARGS=""
OF_TESTPLAN_ARGS=""
OF_SPECDIR_ARGS="--specdir $OF_DEFAULT_SPECDIR"
OF_OUTPUT_FILE="$FUEGO_ENGINE_PATH/work/prolog.sh"
OF_OUTPUT_FILE_ARGS="--output $OF_OUTPUT_FILE"
OF_DISTRIB_FILE=""

```

```

OF_OVGEN="$FUEGO_SCRIPTS_PATH/ovgen/ovgen.py"

function LinuxST_set_overlay_vars() {
    echo "board overlay: $BOARD_OVERLAY"

    if [ "$BOARD_OVERLAY" ]
    then
        echo "using $BOARD_OVERLAY board overlay"

        OF_BOARD_FILE="$OF_ROOT/$BOARD_OVERLAY"

        if [ ! -f $OF_BOARD_FILE ]
        then
            abort_job "$OF_BOARD_FILE does not exist"
        fi

    else
        abort_job "BOARD_OVERLAY is not defined"
    fi

    # check for $DISTRIB and make file path to it
    if [ "$DISTRIB" ]
    then
        echo "using $DISTRIB overlay"

        OF_DISTRIB_FILE="$OF_ROOT/$DISTRIB"

        if [ ! -f $OF_DISTRIB_FILE ]
        then
            abort_job "$OF_DISTRIB_FILE does not exist"
        fi
    else
        abort_job "DISTRIB is not defined"
    fi

    if [ "$BATCH_TESTPLAN" ]
    then
        echo "using $BATCH_TESTPLAN batch testplan"
        OF_TESTPLAN="$OF_ROOT/$BATCH_TESTPLAN"

        if [ ! -f $OF_TESTPLAN ]
        then
            abort_job "$OF_TESTPLAN does not exist"
        fi

        OF_TESTPLAN_ARGS="--testplan $OF_TESTPLAN"

    elif [ "$TESTPLAN" -ne "" ]
    then
        echo "BATCH_TESTPLAN is not sent, using $TESTPLAN testplan"
        OF_TESTPLAN="$OF_ROOT/$TESTPLAN"

        if [ ! -f $OF_TESTPLAN ]
        then
            #abort_job "$OF_TESTPLAN does not exist"
            echo "$OF_TESTPLAN does not exist"
        fi

        OF_TESTPLAN_ARGS="--testplan $OF_TESTPLAN"
    else
        echo "BATCH_TESTPLAN and TESTPLAN are not set"
    fi

    rm -f $OF_OUTPUT_FILE

    OF_OVFILES_ARGS="--ovfiles $OF_DISTRIB_FILE $OF_BOARD_FILE"

    run_python $OF_OVGEN $OF_CLASSDIR_ARGS $OF_OVFILES_ARGS $OF_TESTPLAN_ARGS $OF_SPECDIR_ARGS
    $OF_OUTPUT_FILE_ARGS || abort_job "Error while prolog.sh file generation"

    if [ ! -f "$OF_OUTPUT_FILE" ]
    then
        abort_job "$OF_OUTPUT_FILE not found"
    fi

    source $OF_OUTPUT_FILE
}

```

- Source: /home/jenkins/scripts/LinuxST\_functions.sh
- Purpose: Based on /home/jenkins/scripts/functions.sh, it needs to modify `pre_test` function to LinuxST\_pre\_test function; LinuxST\_pre\_test function will initial the specified working directory for Linux ST
- Modification:

```
function LinuxST_pre_test {
# $1 - tarball template
# Make sure the target is alive, and prepare workspace for the test

export SSHPASS=$PASSWORD
is_empty $1

# Target cleanup flag check
[ "$Target_Cleanup" = "true" ] && target_cleanup $1 || true

cmd "true" || abort_job "Cannot connect to $DEVICE via $TRANSPORT_SSH"

# It is needed to create directory for test logs and system logs on Host PC
mkdir -p $FUEGO_LOGS_PATH/$JOB_NAME/testlogs
mkdir -p $FUEGO_LOGS_PATH/$JOB_NAME/systemlogs
mkdir -p $FUEGO_LOGS_PATH/$JOB_NAME/devlogs

# /tmp/${1} is needed to save logs on different partition

# Get target device firmware.
firmware
cmd "echo \"Firmware revision:\" $FWVER" || abort_job "Error while ROOTFS_FWVER command execution on target"

# XXX: Sync date/time between target device and framework host
# Also log memory and disk status as well as non-kernel processes, and interrupts

ov_rootfs_state

# Create workspace on Board
cmd "rm -rf $FUEGO_HOME/fuego.$1 /tmp/$1; mkdir -p $FUEGO_HOME/fuego.$1 /tmp/fuego.$1" ||
abort_job "Could not create $1 and /tmp/$1 on $DEVICE"
#cmd "rm -rf $FUEGO_HOME/fuego.$1; mkdir -p $FUEGO_HOME/fuego.$1" || abort_job "Could not
create $1 on $DEVICE"

# Log test name
ov_logger "Starting test ${JOB_NAME}"

# flush buffers to physical media and drop filesystem caches to make system load more predictable
during test execution
ov_rootfs_sync

ov_rootfs_drop_caches
}

#####
# Function name: LinuxST_compress_tarball
# Feature: Compress tarball (Eg: device.tar.gz) to deploy test script onto target device
# Argument : None
# Return value: Returns 1 if compress fail; 0 - successful.
#####
function LinuxST_compress_tarball {
echo "Compress test package"

# Browse to test case location
cd ${FUEGO_TESTS_PATH}/${JOB_NAME}/

# Compress command
tar zcvf ${tarball} device/ 2> /dev/null

# Check returned result of compress command
if [ $? -eq 1 ] ; then
echo "Cannot compress tarball : ${tarball}"
return 1
fi
return 0
}
}
```

### 2.3.4 Test Log Structure

```
[LINUXST-START] Start Testing
<Test log>
[LINUXST-END] End Testing
[LINUXST-RESULT-OK]
```

Description: To help parser module be able to just test result easier, the test log structure is standardized with some prefix tags which show the test progress.

- [LINUXST-START]: Indicate the starting point of a test case
- [LINUXST-END]: Indicate the ending point of a test case
- [LINUXST-RESULT-OK]: Indicate the test result is “pass”

Parser will use these indicators to judge the test result.

### 2.3.5 Functions of Linux System Test

- Source: /home/jenkins/scripts/LinuxST\_functions.sh
- Description:
  - Implement some functions to support Linux System Test such as: mount SD-Card, mount SD-Card
  - Based on structure of /home/jenkins/scripts/functions.sh

```
function LinuxST_set_IP_Address ()
{
    echo "[LinuxST_functions.sh] Set IP Address for platform"

    # Set IPADDR IP address for the board
    cmd "busybox ifconfig eth0 $IPADDR netmask $NETMASK broadcast $BROADCAST up"
}

function LinuxST_clear_FTP_incoming ()
{
    echo "[LinuxST_functions.sh] Clear FTP INCOMING folder on Host PC"

    # remove old data in FTP Directory
    rm -rf ${FTP_DIR}/INCOMING
    sync

    # Create new FTP Directory
    mkdir ${FTP_DIR}/INCOMING

    # Change permission (write) for FTP Directory
    chmod 777 ${FTP_DIR}/INCOMING
    sync
}

function LinuxST_mount_SDCard ()
{
    echo "[LinuxST_functions.sh] Mount SDCard"

    # Create and umount mount point SDCARD_MP_LINUX for mounting preparation
    cmd "mkdir -p ${SDCARD_MP_LINUX} 2>/dev/null"
    cmd "umount -f ${SDCARD_MP_LINUX} 2>/dev/null"

    # Mounting mount point SDCARD_MP_LINUX with device note EMMC_DEV_LINUX formatted "auto"
    cmd "mount -t auto ${SDCARD_DEV_LINUX} ${SDCARD_MP_LINUX}"
}
```

```

function LinuxST_mount_NFS ()
{
    echo "[LinuxST_functions.sh] Mount NFS"

    # Create and umount mount point NFS_MP_LINUX for mounting preparation
    cmd "mkdir ${NFS_MP_LINUX} 2>/dev/null"
    cmd "umount ${NFS_MP_LINUX} 2>/dev/null"

    # Set IPADDR IP address for the board
    cmd "busybox ifconfig eth0 $IPADDR netmask $NETMASK broadcast $BROADCAST up"

    # Mounting mount point NFS_MP_LINUX with tftpboot folder on Linux Host PC
    cmd "busybox mount -t nfs -o nolock,proto=tcp ${IPADDR_HOST}:/tftpboot/ ${NFS_MP_LINUX}"
}

function LinuxST_mount_eMMC ()
{
    echo "[LinuxST_functions.sh] Mount $1"

    # Create and umount mount point EMMC_MP_LINUX for mounting preparation
    cmd "mkdir -p ${EMMC_MP_LINUX} 2>/dev/null"
    cmd "umount -f ${EMMC_MP_LINUX} 2>/dev/null"

    # Mounting mount point EMMC_MP_LINUX with device note EMMC_DEV_LINUX formatted "auto"
    cmd "mount -t auto ${EMMC_DEV_LINUX} ${EMMC_MP_LINUX}"
}

function LinuxST_mount_PCIE ()
{
    echo "[LinuxST_functions.sh] Checking PCIe network interface card"
    cmd "ls /sys/class/net/ | grep ${PCIE_eth} > /dev/null"
    if [ $? == 1 ]; then
        echo "[LinuxST_functions.sh] PCIe network interface card cannot be found"
        return 1
    fi

    echo "[LinuxST_functions.sh] PCIe network interface card was DETECTED"
    echo "[LinuxST_functions.sh] Mount PCIe"

    # Create and umount mount point PCIE_MP_LINUX for mounting preparation
    cmd "mkdir -p ${PCIE_MP_LINUX} 2>/dev/null"
    cmd "umount -f ${PCIE_MP_LINUX} 2>/dev/null"

    # Set IPADDR IP address for the board
    cmd "busybox ifconfig -a ${PCIE_eth} ${PCIE_IPADDR} netmask ${NETMASK} broadcast ${BROADCAST} up"

    # Mounting mount point PCIE_MP_LINUX with tftpboot folder on Linux Host PC
    cmd "busybox mount -t nfs -o nolock,proto=tcp ${PCIE_IPADDR_HOST}:/tftpboot/ ${PCIE_MP_LINUX}"
}

function LinuxST_mount_SATA ()
{
    # $1: mount point
    # $2: /dev/block/...
    echo "[LinuxST_functions.sh] Mount $1"

    # Create and umount mount point $1 for mounting preparation
    cmd "mkdir $1 2>/dev/null"
    cmd "umount $1 2>/dev/null"

    # Mounting mount point $1 with SATA device note $2 formatted "auto"
    cmd "mount -t auto $2 $1"
    #####
}

function LinuxST_umount ()
{
    echo "[LinuxST_functions.sh] Unmounting devices ($1)"
    cmd "umount $1"
}

```

```

function LinuxST_create_bin_file ()
{
    echo "[LinuxST_functions.sh] Preparing binary file"

    FILE_NAME=${DEFAULT_FILE_NAME}
    BLOCKS=${DEFAULT_BLOCKS}
    BYTES=${DEFAULT_BYTES}

    # Set default information
    if [ ! -z "$2" ]; then
        FILE_NAME=$2
    fi
    if [ ! -z "$3" ]; then
        BLOCKS=$3
    fi
    if [ ! -z "$4" ]; then
        BYTES=$4
    fi

    echo $FILE_NAME

    # Create folder containing binary files
    cmd "mkdir ${1} 2>/dev/null"

    # Create FILE_NAME file with capability = BLOCKS*BYTES (bytes)
    cmd "dd if=/dev/zero of=${1}/${FILE_NAME} count=$BLOCKS bs=$BYTES"
    sync
    echo "[LinuxST_functions.sh] LinuxST_create_bin_file is DONE"
}

function LinuxST_gst_setting_hostPC ()
{
    echo "[LinuxST_functions] Apply setting on HostPC to play multimedia data by GStreamer."
    export XDG_RUNTIME_DIR="/run/user/1000"
    export DISPLAY=":0"
    echo "[LinuxST_functions] Play video via GST"
    sleep 2
}

function LinuxST_gst_play_hostPC ()
{
    if [ $# -ne 2 ]; then
        echo "[Linux_functions.sh] Lack of input" >&2
        exit 1
    fi

    STREAM_FORMAT=$1
    STREAM_PATH=$2

    # Setting for PC before play multimedia
    LinuxST_gst_setting_hostPC

    case "$STREAM_FORMAT" in
        "h264" )
            echo "[Linux_functions.sh] Play H.264 video by GStreamer on Host PC"
            ${GST_LAUNCH} filesrc location=${STREAM_PATH} ! h264parse ! avdec_h264 ! autovideosink
            ;;
        "acc" | "AAC" )
            echo "[Linux_functions.sh] Play AAC audio by GStreamer on Host PC"
            ${GST_LAUNCH} filesrc location=${STREAM_PATH} ! decodebin ! alsasink
            ;;
        "mp4" | "MP4" )
            echo "[Linux_functions.sh] Play MP4 audio by GStreamer on Host PC"
            ${GST_LAUNCH} filesrc location=${STREAM_PATH} ! decodebin ! alsasink
            ;;
        # Terminate test case if input is wrong or unsupported format
        *)
            echo "[Linux_functions.sh] Input format is NOT supported by this function."
            exit 1
            ;;
    esac
}

```

```

function LinuxST_check_medaiinfo ()
{
    # Checking input argument
    if [ $# -ne 3 ]; then
        echo "[Linux_functions.sh] Usage: LinuxST_check_medaiinfo {check_type} {mode} {STREAM}" >&2
        echo "[Linux_functions.sh] {check_type}: 'encode' or 'record'" >&2
        echo "[Linux_functions.sh] {mode}: 'show' or 'compare'" >&2
        echo "[Linux_functions.sh] {STREAM}: multimedia data which is need to check" >&2
        exit 1
    fi

    # Define variables
    check_type=$1
    mode=$2
    STREAM=$3
    log="mediainfo.log"

    # Use Mediainfo tool to parse specification of multimedia data to log file
    /usr/bin/mediainfo $STREAM > ${log}

    case ${check_type} in
        video )
            # Specification of encoded video
            # Get profile
            profile_tmp=`cat ${log} | grep "Format profile" | grep "@ "`
            profile_tmp=${profile_tmp#*: }
            profile_tmp1=`echo ${profile_tmp} | cut -d '@' -f 1`
            case ${profile_tmp1} in
                Baseline )
                    profile=bp
                    ;;
                Main )
                    profile=mp
                    ;;
                High )
                    profile=hp
                    ;;
            esac
            # Get level
            lv=`echo ${profile_tmp} | cut -d '@' -f 2 | grep -o "[0-9]" | tr -d '\n'`
            case ${lv} in
                1b )
                    lv=1.b
                    ;;
                1 | 2 | 3 | 4 | 5 )
                    lv=${lv}.0"
                    ;;
            esac
            # Get resolution
            height=`cat ${log} | grep "Height" | cut -d ':' -f 2 | cut -d ' ' -f 2`
            width=`cat ${log} | grep "Width" | cut -d ':' -f 2 | cut -d ' ' -f 2`
            res=`echo ${width}x${height}`
            # Get frame rate
            fps_tmp=`cat ${log} | grep "fps"`
            fps_tmp=${fps_tmp#*: }
            fps=${fps_tmp%.*}
            # Get scan type
            scan_type=`cat ${log} | grep "Scan type"`
            scan_type=${scan_type#*: }
            case ${scan_type} in
                Progressive )
                    scan_type="p"
                    ;;
                Interlace )
                    scan_type="i"
                    ;;
            esac
            # Get bit rate
            bps=`cat ${log} | grep "Nominal bit rate" | head -1 | grep -o "[0-9]" | tr -d '\n'`

            # Assign multimedia specification in to spec[] array
            spec[0]=${profile}
            spec[1]=${lv}
            spec[2]=${res}
            spec[3]=`echo ${fps}p`
            spec[4]=`echo ${bps}kbps`
        ;;
    esac
}

```

```

case ${mode} in
    show )
        echo ">>>> $STREAM <<<<"
        echo "Profile@level: ${profile_tmpl}@L${lv}"
        echo "Resolution   : ${height}x${width}"
        echo "Frame rate    : ${fps}${scan_type}"
        echo "Bit rate      : ${bps}kbps"
        ;;
    compare )
        echo "Compare stream name with specification..."
        for i in {0..4}; do
            if [[ "${name[i]}" != "${spec[i]}" ]]; then
                echo "[LinuxST_functions.sh] Mediainfo checked FAILED"
                echo "[LinuxST_functions.sh] [LinuxST_check_mediainfo] Additional debug info:
${name[i]} != ${spec[i]}"
                exit 1
            # else
            # echo "i"
            fi
        done
        echo "[LinuxST_functions.sh] Mediainfo checked PASSED"
        ;;
esac
;;
audio )
    # parse audio specification from input name
    inp_name=`echo ${STREAM} | rev | cut -d '/' -f 1 | rev`
    inp_format=`echo ${inp_name} | cut -d '_' -f 2`
    if [[ "${inp_format}" == "aac" ]]; then
        inp_format="Advanced Audio Codec"
    fi

    inp_ch=`echo ${inp_name} | cut -d '_' -f 3 | cut -d 'c' -f 1`
    inp_sampling_rate=`echo ${inp_name} | cut -d '_' -f 4 | cut -d 'h' -f 1`

    # parse specification from Mediainfo
    format=`cat ${log} | grep "Format/Info" | cut -d ':' -f 2 | cut -d ' ' -f 2-5`
    channel=`cat ${log} | grep "Channel(s)" | cut -d ':' -f 2 | cut -d ' ' -f 2-5 | cut -d ' ' -f 1`
    sampling_rate=`cat ${log} | grep "Sampling rate" | cut -d ':' -f 2 | cut -d ' ' -f 2-5`

    unit_sampling_rate=`echo ${sampling_rate} | cut -d ' ' -f 2`
    if [[ "${unit_sampling_rate}" == "KHz" ]]; then
        tmp_sampling_rate=`echo ${sampling_rate} | cut -d ' ' -f 1`
        sampling_rate=`echo ${tmp_sampling_rate} \* 1000 | bc -l | cut -d '.' -f 1`
    fi

    case ${mode} in
        show )
            echo "Format/Info:      ${format}"
            echo "Number of channel(s): ${channel}"
            echo "Sampling rate:      ${sampling_rate}"
            ;;
        compare )
            # Compare audio format
            if [[ ${inp_format} != ${format} ]]; then
                echo "[Linux_commons.sh] ERROR: Mediainfo check wrong format"
                echo "[Linux_commons.sh] Additional debug info: ${inp_format} != ${format}"
                exit 1
            fi
            # Compare audio channel(s)
            if [[ ${inp_ch} != ${channel} ]]; then
                echo "[Linux_commons.sh] ERROR: Mediainfo check wrong number of channel"
                echo "[Linux_commons.sh] Additional debug info: ${inp_ch} != ${channel}"
                exit 1
            fi
            # Compare audio sampling rate
            if [[ ${inp_sampling_rate} != ${sampling_rate} ]]; then
                echo "[Linux_commons.sh] ERROR: Mediainfo check wrong sampling rate"
                echo "[Linux_commons.sh] Additional debug info: ${inp_sampling_rate} != ${sampling_rate}"
                exit 1
            fi
            echo "[Linux_commons.sh] Mediainfo checked PASS"
            ;;
    esac
;;
esac
}

```



### 2.3.6 New parser

- Source: /home/jenkins/scripts/LinuxST\_functions.sh
- Modification:
  - Add new function to call new parser
  - Get return value from parser to judge test result

```
function LinuxST_fail_check_cases() {

testlog=${FUEGO_LOGS_PATH}/${JOB_NAME}/testlogs/${NODE_NAME}.${BUILD_ID}.${BUILD_NUMBER}.log
    failtoken=${FUEGO_SCRIPTS_PATH}/LinuxST_fail_token.txt

    echo -e "\n RESULT ANALYSIS \n"

    check_result=$(LinuxST_run_python ${FUEGO_SCRIPTS_PATH}/parser/LinuxST_parser.py $testlog $failtoken)

    result_code=${check_result:0:4}
    echo $result_code

    case "$result_code" in
        "pass")
            echo "TEST RESULT : PASSED"
            ;;
        "fail")
            echo "TEST RESULT : FAIL"
            fail_job
            ;;
        "abor")
            echo "TEST RESULT : ABORTED"
            abort_job ""
            ;;
    esac
}
```

- Source: /home/jenkins/scripts/LinuxST\_common.sh
- Modification:
  - Add new python calling function to get test result.
  - Add new function to terminate test when test result is fail

```
function LinuxST_run_python() {
    if [ ! -z $ORIG_PATH ]
    then
        PATH=$ORIG_PATH python "$@"
    else
        python2.7 "$@"
    fi
}

# Set job (test case) is fail
function fail_job {
    set +x
    echo -e "\n*** FAILURE ***\n"

    exit -1
}
```

- Source: /home/jenkins/scripts/LinuxST\_fail\_token.txt
- Description:
  - LinuxST\_fail\_token.txt is added newly to FUEGO.
  - Test result is “fail” if log file contains messages that are existed in LinuxST\_fail\_token.txt

```
Error
No such file or directory
Kernel panic
Aborted
```

For example:

```
[LINUXST-START] Start Testing
write test for /mnt/sdcard (bs=100K
count=1)
/mnt/sdcard : No such file or directory
[LINUXST-END] End Testing
```

Test log

```
Error
No such file or directory
Kernel panic
```

LinuxST\_fail\_token.txt

⇒ Test result is "FAIL"

- Source: /home/jenkins/scripts/parser/LinuxST\_parser.py
- Description:
  - Parser reads Fail Tokens which are stored at /home/jenkins/scripts/LinuxST\_fail\_token.txt to check log file
  - Parser return test result to LinuxST\_fail\_check\_cases function

```
#!/bin/python

import re, sys

start=0
end=0
result=-1
logfile_name = sys.argv[1]
fail_token_name = sys.argv[2]

log_file = open(logfile_name, 'r')
fail_sample = open(fail_token_name, 'r')

line_log = log_file.readlines()
line_fail_sample = fail_sample.readlines()

for cur_line_log in range(len(line_log)):
    for cur_line_fail in range(len(line_fail_sample)): # Search FAIL token
        if re.search(line_fail_sample[cur_line_fail].rstrip('\n'), line_log[cur_line_log],
re.IGNORECASE): # Search FAIL token
            print 'fail - Error message: line: ' + str(cur_line_log) + ' - Message: ' +
line_log[cur_line_log]
            result = 0
            if re.search('LINUXST-START', line_log[cur_line_log]): # Search START TESTING token
                start = 1
            if re.search('LINUXST-END', line_log[cur_line_log]): # Search END TESTING token
                end = 1
            if (re.search('LINUXST-RESULT-OK', line_log[cur_line_log])) and (result != 0): # Search
TESTING RESULT token
                result = 1

if start == 1:
    if end == 1:
        if result == 1:
            print 'pass'
        else:
            print 'fail'
    else:
        print 'fail'
else:
    print 'abort'

log_file.close()
fail_sample.close()

sys.exit()
```

## 3 Hardware reset implementation for FUEGO

### 3.1 Background

Original FUEGO resets board by command:

```
function LinuxST target_reboot {
    export SSHPASS=$PASSWORD
    cmd "/sbin/reboot"
    ...
}
```

In case the board hangs up during execution, the board cannot be reset by command. Therefore, need to implement the Hardware reset for FUEGO to prevent this issue.

## 3.2 Overview of Hardware reset control

### 3.2.1 Component of Hardware reset board

- Components of Hardware reset board are described as below:

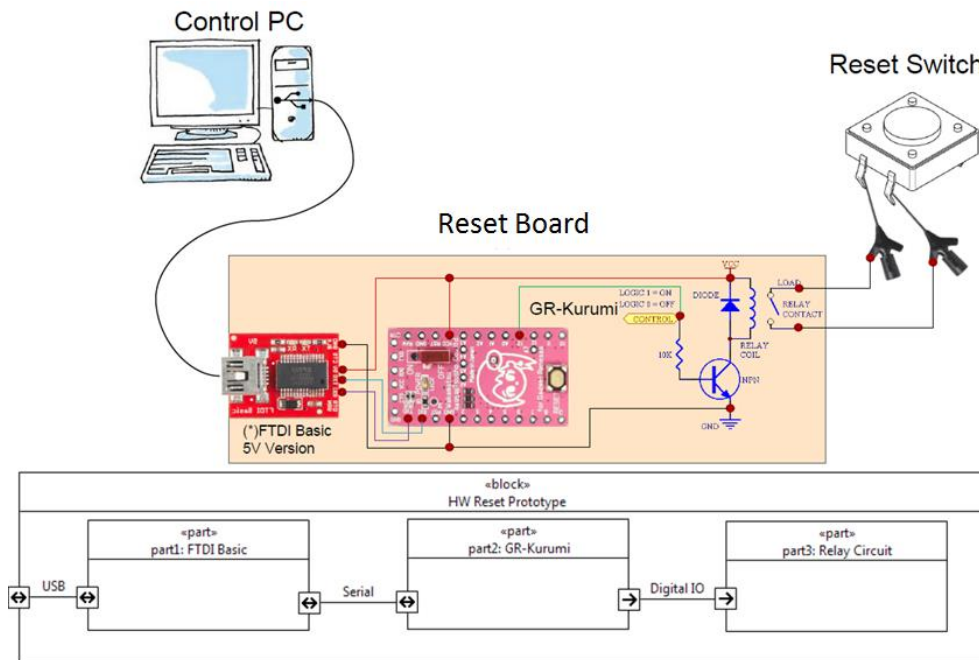


Figure 13: Components of HW reset board

- Configuration of Reset\_board:
  - Serial port: 8-N1
  - Baud rate: 38400
  - Serial message for reset activation request: \$c1
- When resetting Target Board by Hardware, the reset message will be sent by application (r\_hwreset\_util) from Host PC to reset board and this board will send a reset signal to Target board.

### 3.2.2 Reset activate sequence

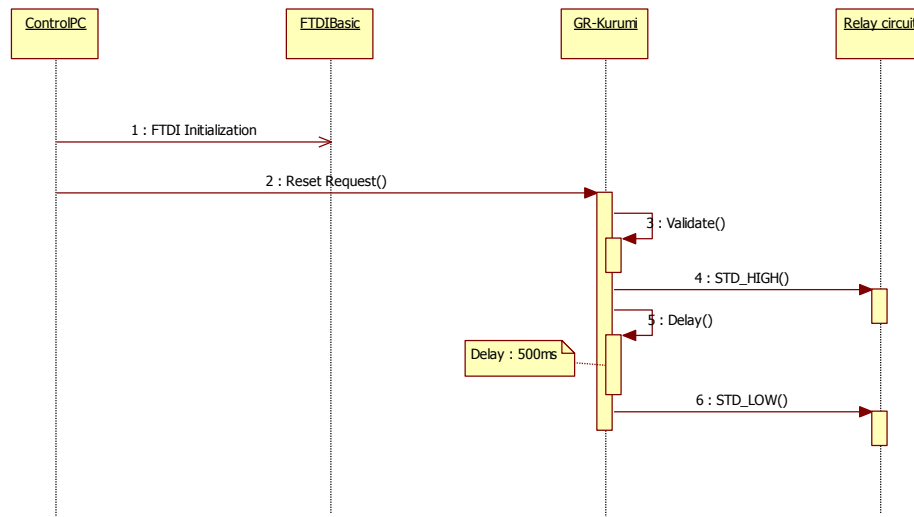


Figure 14: Sequence diagram of hardware reset

## 3.3 Modification to implement Hardware reset to FUEGO

### 3.3.1 Block Diagram

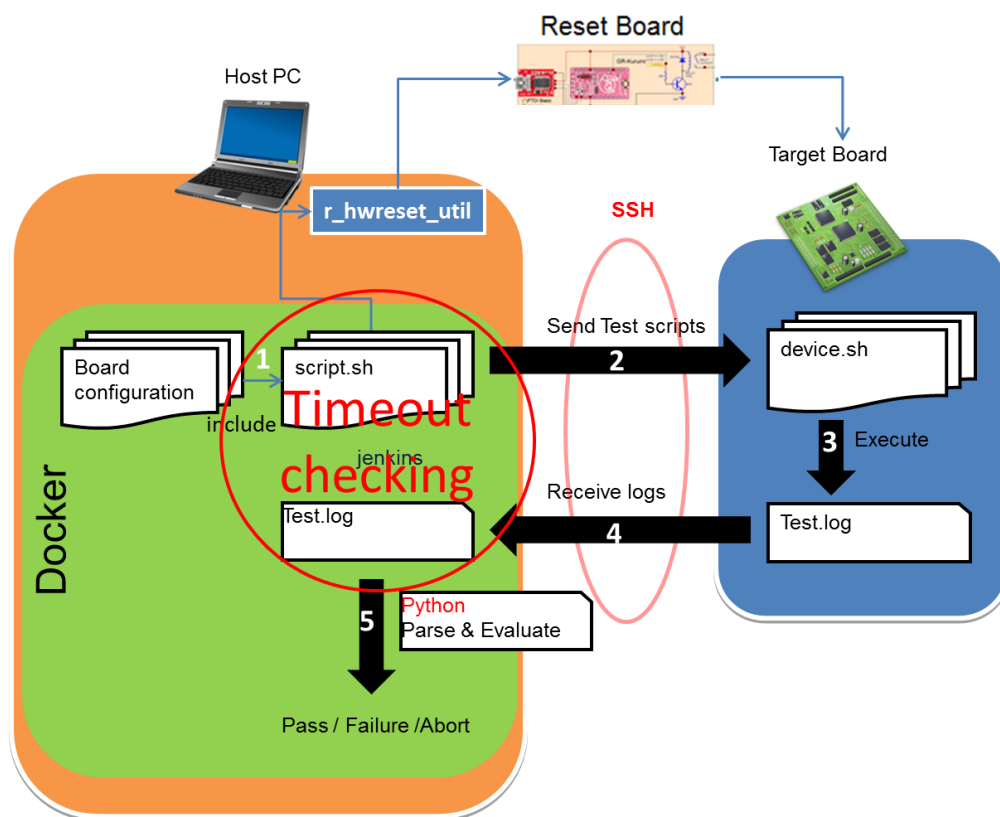


Figure 15: Block diagram of Linux FUEGO after implement Hardware reset

Explanation:

- Hardware reset:
  - In original, FUEGO already has a function (**LinuxST\_reset**) which can reset Target board by command.
  - To implement Hardware reset, need to modify the function to support HW reset which resets board by calling **r\_hwreset\_util**.
  - Target board will be reset by **SW or HW** base on the configuration in file `/home/jenkins/overlays/boards/hwreset.config`.
- Time out checking:
  - Below is the flow chart of timeout handler for FUEGO Linux ST:



Figure 16: Flow chart of timeout handler for FUEGO Linux ST

- While test case is executing, the **timeout checking** will check whether test case executes over expected time or not.
  - If test case is finished, it breaks the while loop.
  - If test case is not finished, hang up checking will check whether test case executes is hang up or not each several times, or waiting time\_Checkhangup pass. (e.g. time\_Checkhangup = 30s) by sending a command to Target board and get the response value.
  - In this situation, time\_Checkhangup is 30s because :
    - 30s is not too sort to effect the performance of Target Board ( send many command to Target board in a sort time will make performance of board go down).
    - And 30s is not too long to waste time for waiting when hang up in Target board occurs.

- After over the timeout duration, hang up checking will check again to check whether test case executes is hang up or not. If not, test case is time out.
- If hang up occurs:
  - The result of test case will be judged as Fail, the reason is “Board hangs up”.
  - There is no test log, so a test log is created with result and reason of the test case.
  - Then, test case will be killed (kill test\_run in Host PC but in Target board, test case is still running). After that, if user runs multi-test cases (batch run) FUEGO will move to execute the next test case, Target board will be reset.
- If timeout occurs:
  - The result of test case will be judged as Fail, the reason is “Timeout”.
  - In case time out, FUEGO could get log from Target board and the reason of fail case (timeout) will be recorded in this log (test case’s test log).
- Then, test case will be killed (kill test\_run in Host PC but in Target board, test case is still running). After that, if user runs multi-test cases (batch run) FUEGO will move to execute the next test case, Target board will be reset.
- The timeout checking has some point need to be noticed as below:
  - The expected execution time for each test case is defined by developer.

$\text{TIMEOUT} = \text{DELTA\_TIMEOUT} + \text{TIMEOUT\_BASE}$

▪ **DELTA\_TIMEOUT:**

- It is timeout value of each Target board which is defined based on performance of each Target board. (Define in configuration file of Target board, example: /home/jenkins/overlays/boards/renesas.board.salvator-X).
- Currently, TIMEOUT\_BASE value was defined base on Lager board, so DELTA\_TIMEOUT of Lager board is 0. If performance of Target board is lower than Lager board, increase DELTA\_TIMEOUT and reverse (this value can be negative number).

▪ **TIMEOUT\_BASE:** Time out value for each kind of test case.

- Example: Executing test case LinuxST.CopyLargeFiles.NFS.SDCard on Koelsch board:
- DELTA\_TIMEOUT: Koelsch board has lower performance than Lager board, so this value should be increase. Eg: DELTA\_TIMEOUT= 150 (this number is an example, not exactly)
- TIMEOUT\_BASE of this test case is 33 min.

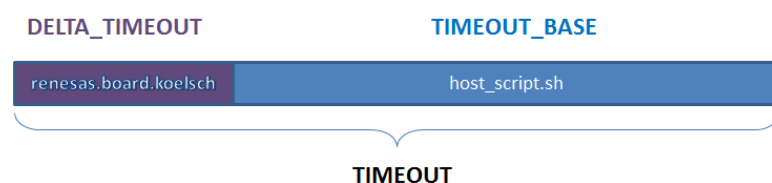


Figure 17: Elements of Timeout

- TIMEOUT in this situation is  $33 \times 60 + 150 = 2130(s) \sim 36 \text{ min}$  If test case is not defined its own TIMEOUT\_BASE, FUEGO will use the default TIMEOUT\_BASE value (3600s).

### 3.3.2 Modification Diagram

Below is the comparison (new, modify, original) between a diagram of Linux FUEGO (Figure 11: Workflow AFTER modification) and a diagram of Linux FUEGO after implement the Hardware reset

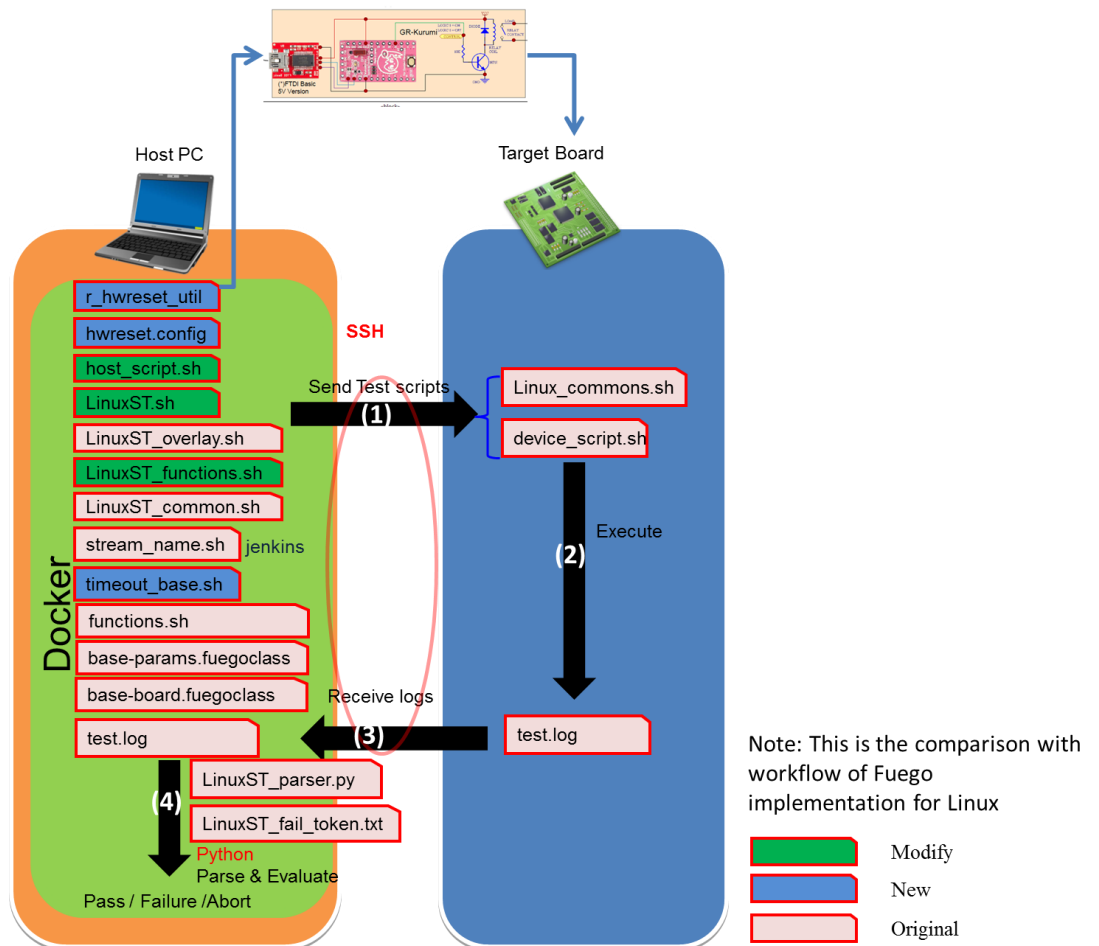


Figure 18: Workflow of Linux FUEGO after implement the Hardware reset

- New files:
  - timeout\_base.sh: Define time out value for each kind of test case.
  - r\_hwreset\_util: The application use to control Reset Board from Host PC.
  - hwreset.config: Define the attendance of Hardware reset and device node of Reset Board.
- Modified files: The below files are modified:
  - LinuxST.sh: Add hang up checking.
  - LinuxST\_functions.sh: Add function to handle a test case's timeout or hang up and function to support Hardware reset.
  - host\_script.sh: Add time out for each test case.

### 3.3.3 Sequence Diagram

The sequence diagram of Hardware reset and timeout handler proceed from Step 37 – 45 in sequence diagram of implementation specification of FUEGO for Linux ST as below:

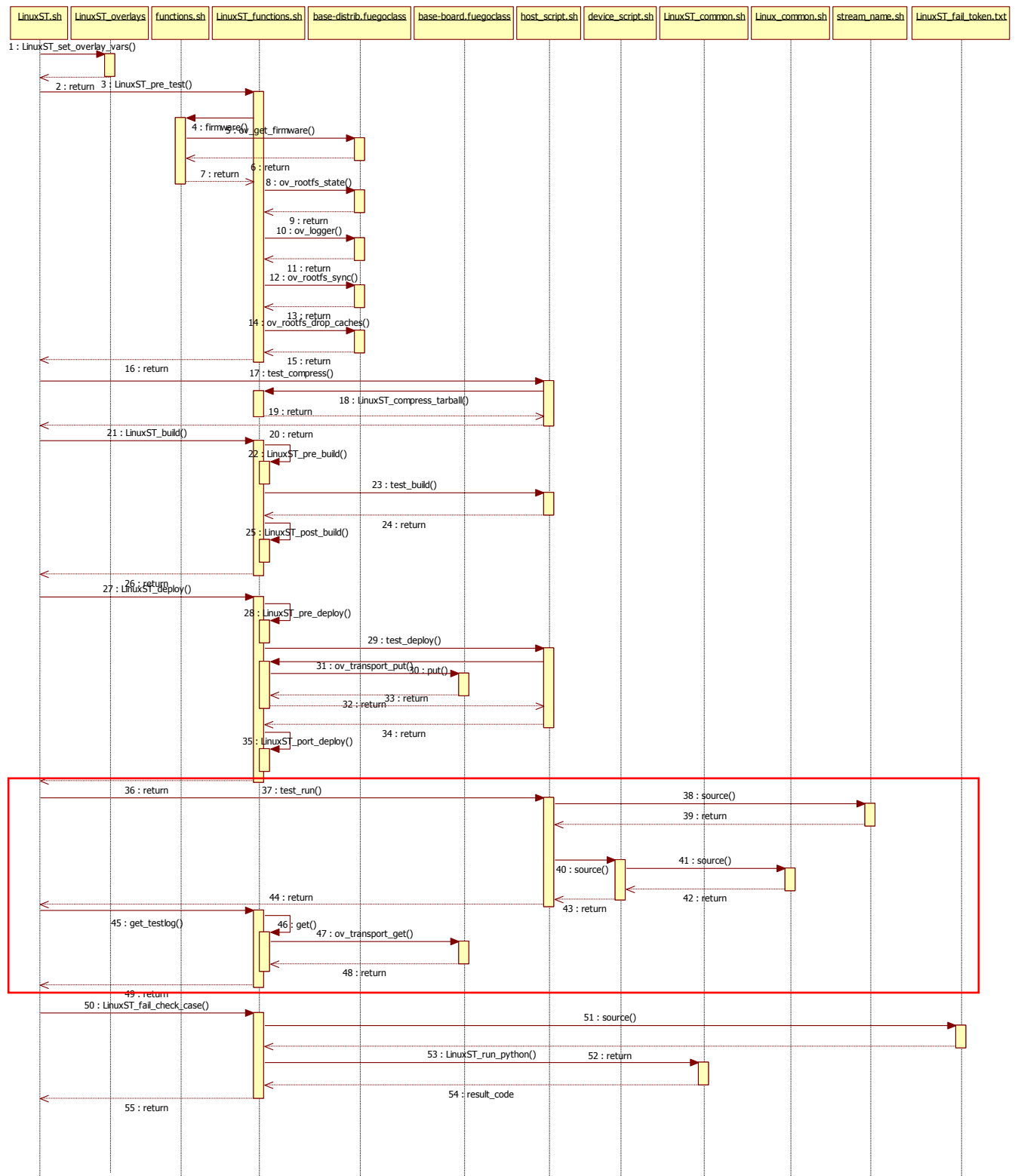


Figure 19: Sequence Diagram of FUEGO for Linux ST



The sequence diagram of Hardware reset and timeout handler (Inside of the red rectangle in Figure 21):

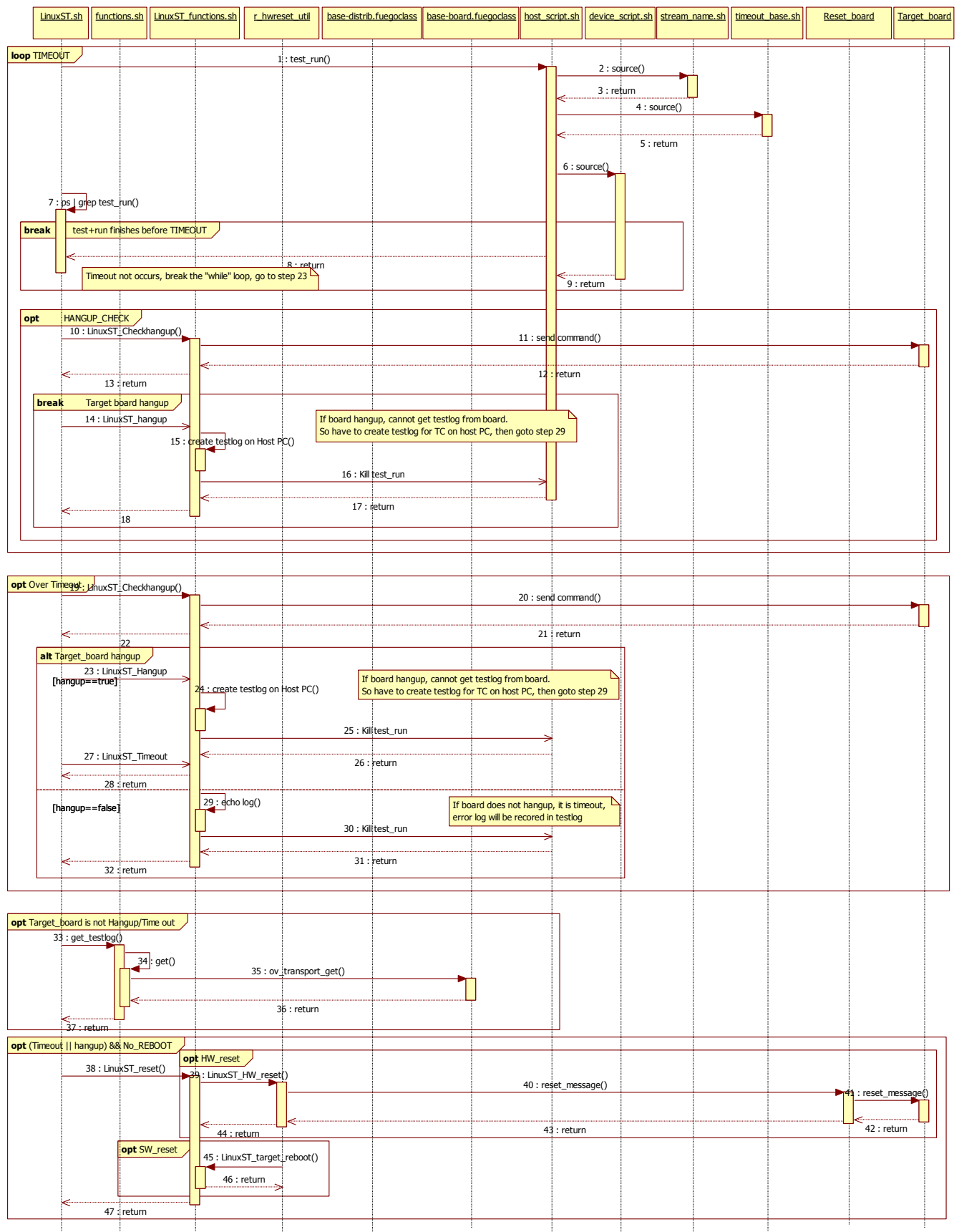


Figure 20: Sequence Diagram of Hardware reset and timeout handler for FUEGO Linux ST

Explanation:

(3) While test\_run is executing, a “while” loop will run until:

- The execution time of test case is over the TIMEOUT (it means time out occurs)

Or

- The test\_run is finished before TIMEOUT (it means time out not occurs), it will break the “while loop” and go to step (33) get\_testlog.

Or

- The test\_run is hang up before TIMEOUT or test case finish, it will handle the hang up event, then break the “while loop” and go to step (38) to reset HW.

(7) Check the existence of test\_run every second.

(10) – (18) To check hang up occurs every several time (waiting time to check hang up), LinuxST\_Checkhangup will be called. In this function, it will:

- Send a command to Target board:
- If Target board can finish the command, it means test case is not hang up.
- If Target board cannot finish the command, it means the Target board is hanged up. In this case, cannot get test log from board. So, LinuxST\_Hangup is called to create a fake of test log and announce that test case fails because the Target board hangs up in this test log. After that, kill test\_run on Host PC.

(19) – (32) After over TIMEOUT, but test case is not finished, LinuxST\_Checkhangup is called to check whether Target board is hang up or not (to prevent Target Board hang up in the last second). If not, test case is timeout.

- Send a command to Target board:
- If Target board can finish the command, it means test case is not hang up but time out. In this case, test result is failed and announcement that test case is failed because of time out is record to test log.
- If Target board cannot finish the command, it means the Target board is hanged up. FUEGO will handle it like hang up case as above.
- After check Target board statement, kill test\_run on Host PC.

(34) – (43) In case test case get time out or Target board hangs up, but user does not choose REBOOT before running test, Target board will be force reset by Hardware or Software base on configuration in /home/jenkins/overlays/boards/hwreset.config.

## 3.4 Detail modification

### 3.4.1 Main script

- Source: /home/jenkins/scripts/LinuxST.sh
- Purpose:
  - Add time out checking to handle 2 cases which a test case get time out or a board hangs up.
  - Support to reset the Target board by Reset board.
- Modification:

```
source $FUEGO_SCRIPTS_PATH/LinuxST_overlays.sh
LinuxST_set_overlay_vars

source $FUEGO_SCRIPTS_PATH/functions.sh
source $FUEGO_SCRIPTS_PATH/LinuxST_functions.sh
source $FUEGO_SCRIPTS_PATH/reports.sh
source $FUEGO_SCRIPTS_PATH/timeout_base.sh
testlog=${FUEGO_LOGS_PATH}/${JOB_NAME}/testlogs/${NODE_NAME}.${BUILD_ID}.${BUILD_NUMBER}.log
# Target board will be reboot if option REBOOT is chosen
if $REBOOT
then
    LinuxST_reset
else
    echo "no reset"
    NO_REBOOT=1
fi

LinuxST_pre_test $TESTDIR

test_compress

if $Rebuild; then
    LinuxST_build
fi

LinuxST_deploy

##### Start timeout checking HERE #####
echo "[OSSMM] Start timeout checking"
echo "$TIMEOUT_BASE"
if [ "${TIMEOUT_BASE}" == "" ]
then
    echo "TIMEOUT_BASE is not defined, use value default : 3600s "
    TIMEOUT_BASE=3600
fi

# DELTA TIMEOUT: Time out value of each target board (in /home/jenkins/overlays/boards) which
is defined base on performance of each board
let TIMEOUT=DELTA_TIMEOUT+TIMEOUT_BASE
echo ${TIMEOUT}

TIMEOUT_FLG=0
# Record start time of test case
Start_Time=$(date +%s")
# Execute Test Script on Target Board
test_run &
test_pid=$!

echo "[OSSMM] start while loop"
# This argument to count until TIMEOUT_LinuxST_Checkhangup to check hang up
SECONDS=0
# Start while loop to check time out
while [ $TIMEOUT_FLG -le $TIMEOUT ]
do
    let TIMEOUT_FLG=TIMEOUT_FLG+1
    # Check the existence of $test_pid, if it does not exist, test case finished
    ps | grep $test_pid
    ret=$?
```

```

if [ "${ret}" != "0" ]
then
    echo "[OSSMM] finished test case"
    break
fi
sleep 1
# FUEGO check hang up every 'TIMEOUT_LinuxST_Checkhangup' seconds
if [ $SECONDS == $time_Checkhangup ]
then
    echo "[OSSMM] check hang up"
    LinuxST_Checkhangup
    if [ "${hang_up}" == "0" ]; then
        echo "board ok"
    else
        echo "board hang up"
        LinuxST_Hangup
    fi
    # When LinuxST_Checkhangup running, it sleep about 'TIMEOUT_LinuxST_Waitting' seconds,
    so need to update TIMEOUT_FLG
    let TIMEOUT_FLG=TIMEOUT_FLG+TIMEOUT_LinuxST_Waitting
    SECONDS=0
fi

done

echo "[OSSMM] end while loop"

echo $TIMEOUT_FLG

# Execution time over TIMEOUT, need to check in the last
if [ $TIMEOUT_FLG -ge $TIMEOUT ] && [ "${hang_up}" != "1" ];
then
    LinuxST_Checkhangup
    if [ "${hang_up}" == "0" ]; then
        echo "board time out"
        LinuxST_Timeout
    else
        echo "board hang up"
        LinuxST_Hangup
    fi
fi

##### End timeout checking #####
# Record end time of test case
End_time=$(date +%s")
# set_testres_file
set_testres_file
# Caculate run time of test case
runtime=$((End_time-Start_Time))
# Copy test log from Target Board to Host PC. If board hang up, cannot get log from board.
if [ "${hang_up}" != "1" ]
then
    get_testlog $TESTDIR
fi

# Convert execution time of test case to format hh:mm:ss, and record it in to test log of test
case.
runtime=$(python -c "print '%02u:%02u:%02u' % ((${runtime})/3600, (${runtime})%3600/60,
(${runtime})%60)")
echo "Duration : ${runtime} " >> $testlog

if [ "${time_out}" == "1" ]
then
    echo "Error occur" >> $testlog
    echo "Time Out" >> $testlog
fi

# Incase Target board timeout or hang up, but user does not choose REBOOT, force reboot Target
board.
if [[ "${time_out}" == "1" || "${hang_up}" == "1" ]] && [ "${NO_REBOOT}" == "1" ]
then
    echo "reset"
    LinuxST_reset
fi

# Check Test Log to judge Test Result
LinuxST_fail_check_cases

```

### 3.4.2 Other scripts

- Source: /home/jenkins/scripts/LinuxST\_function.sh
- Purpose:
  - Add function LinuxST\_Checkhangup
  - Add function LinuxST\_Hangup
  - Add function LinuxST\_Timeout
  - Add function LinuxST\_reset
  - Add function LinuxST\_HWreset
- Modification:

```
...
. $FUEGO_SCRIPTS_PATH/timeout_base.sh
REBOOT_TIME=45 # This magic number required as we need to wait until device reboots
...
#####
# Function name : LinuxST_Checkhangup
# Feature : Send command to board, get response of board to judge board hang up or not
#           hang_up=0: board not hang up
#           hang_up=1: board hang up
# Argument : None
# Return value : None
#####
function LinuxST_Checkhangup {

# Send command to board to test whether board hang up or not
cmd "echo "test"" &
# Get PID of above command
cmd_pid=$!
# wait for 5s
sleep TIMEOUT_LinuxST_Waitting
# Check existence of command
ps | grep $cmd_pid
ret_cmd=$?
if [ "${ret_cmd}" != "0" ]
then
# board is still alive, TC got time out
hang_up=0
else
# kill process cmd echo
echo "kill cmd echo"
kill $cmd_pid
hang_up=1
fi
}
#####
# Function name : LinuxST_Hangup
# Feature : Handle in case board hang up
# Argument : None
# Return value : None
#####
function LinuxST_Hangup {

testlog=${FUEGO_LOGS_PATH}/${JOB_NAME}/testlogs/${NODE_NAME}.${BUILD_ID}.${BUILD_NUMBER}.log

# Handle in case Board hang up. Cannot get log from board to judge, so create a fake log which
announce TCs fail due to board hang up
echo $testlog
echo "[LINUXST-START] Start Testing" > $testlog
echo "error occur" >> $testlog
echo "Board hang up" >> $testlog
echo "[LINUXST-END] End Testing" >> $testlog
# Kill process test run
echo "kill test run"
kill $test_pid
ret=$?
echo $ret
if [ "${ret}" != "0" ]
then
```

```

        echo "kill process fail"
        echo "TC fail and reset board"
        echo "Kill process test run fail" >> $testlog
    fi
}

#####
# Function name : LinuxST_Timeout
# Feature      : Handle TC when it got time out ( time out of each test case is defined in timeout_base.sh)
# Argument     : None
# Return value  : None
#####

function LinuxST_Timeout {

testlog=${FUEGO_LOGS_PATH}/${JOB_NAME}/testlogs/${NODE_NAME}.${BUILD_ID}.${BUILD_NUMBER}.log
time_out=1
# Kill process test run
echo "kill test run"
kill $test_pid
ret=$?
echo $ret
if [ "${ret}" != "0" ]
then
    echo "kill process fail"
    echo "TC fail and reset board"
    echo "Kill process test run fail" >> $testlog
fi
}

#####
# Function name : LinuxST_reset
# Feature      : Reset system by hardware or software based on config
# Argument     : None
# Return value  : None
#####

function LinuxST_reset {
. $FUEGO_OVERLAYS_PATH/boards/hwreset.config
echo $HW_RESET
if [ "${HW_RESET}" == "1" ]
then
    echo "HW reset exist"
    echo "Reset Target board by HW"
    LinuxST_HW_reset
else
    echo "Reset Board by SW"
    target_reboot
    sleep 10
fi
}

#####
# Function name : LinuxST_HW_reset
# Feature      : Reset system by hardware
# Argument     : None
# Return value  : None
#####

function LinuxST_HW_reset {

# Go to folder contain source of hw reset
cd $FUEGO_ENGINE_PATH/r_hwreset_util
# Call HW reset.
./r_hwreset_util /dev/${RVC_HW_RESET_DEVICE}
sleep ${REBOOT_TIME} # This magic number required as we need to wait until device reboots
cmd "true"
if [ $? ]; then
    true
else
    false
fi
}

...

```

- Source: /home/jenkins/scripts/timeout\_base.sh
- Purpose:
  - Define timeout base value for each group of test case

```

TIMEOUT_LinuxST_Waitting="5"                # Wait for 5s to check response of
TIMEOUT_LinuxST_Checkhangup="30             board
TIMEOUT_LinuxST_FTP_LargeFile_Board_PC="960" # Wait for 30s to check board hang
TIMEOUT_LinuxST_FTP_LargeFile_PC_Board="960" up
TIMEOUT_LinuxST_FTP_ManyFile_Board_PC="1200" #timeout = 16 min
TIMEOUT_LinuxST_FTP_ManyFile_PC_Board="1200" #timeout = 16 min
TIMEOUT_LinuxST_FTP_Simultaneous_Board_PC="1440" #timeout = 20 min
TIMEOUT_LinuxST_FTP_Simultaneous_PC_Board="7200" #timeout = 20 min
                                                #timeout = 24 min
TIMEOUT_LinuxST_LargeFile_USB20_USB20="3120" #timeout = 2h
TIMEOUT_LinuxST_LargeFile_USB20_OTHERS="1500"
TIMEOUT_LinuxST_LargeFile_OTHERS_USB20="1500" #timeout = 52 min
TIMEOUT_LinuxST_LargeFile_USB30_USB30="3120" #timeout = 25 min
TIMEOUT_LinuxST_LargeFile_USB30_OTHERS="1620" #timeout = 25 min
TIMEOUT_LinuxST_LargeFile_OTHERS_USB30="1500" #timeout = 52 min
                                                #timeout = 27 min
TIMEOUT_LinuxST_LargeFile_OTHERS_NFS="1980"   #timeout = 25 min
TIMEOUT_LinuxST_LargeFile_OTHERS_SDCard="1980"
TIMEOUT_LinuxST_LargeFile_OTHERS_SDLinux="1500" #timeout = 33 min
TIMEOUT_LinuxST_LargeFile_OTHERS_eMMC="1980"  #timeout = 33 min
TIMEOUT_LinuxST_LargeFile_OTHERS_HDD="1080"   #timeout = 25 min
TIMEOUT_LinuxST_LargeFile_DVD_OTHERS="2040"   #timeout = 33 min
                                                #timeout = 18 min
TIMEOUT_LinuxST_ManyFile_USB20_USB20="3720"   #timeout = 34 min
TIMEOUT_LinuxST_ManyFile_USB20_OTHERS="1260"
TIMEOUT_LinuxST_ManyFile_USB30_USB30="3720"   #timeout = 1h2 min
TIMEOUT_LinuxST_ManyFile_USB30_OTHERS="1380"  #timeout = 21 min
                                                #timeout = 1h2 min
TIMEOUT_LinuxST_ManyFile_OTHERS_NFS="1740"   #timeout = 23 min
TIMEOUT_LinuxST_ManyFile_OTHERS_SDCard="2100"
TIMEOUT_LinuxST_ManyFile_OTHERS_SDLinux="1560" #timeout = 29 min
TIMEOUT_LinuxST_ManyFile_OTHERS_eMMC="2100"   #timeout = 35 min
TIMEOUT_LinuxST_ManyFile_OTHERS_HDD="1200"    #timeout = 26 min
TIMEOUT_LinuxST_ManyFile_DVD_OTHERS="2340"    #timeout = 35 min
                                                #timeout = 20 min
TIMEOUT_LinuxST_Simultaneous_USB20_USB20="5640" #timeout = 39 min
TIMEOUT_LinuxST_Simultaneous_USB20_OTHERS="2880"
TIMEOUT_LinuxST_Simultaneous_USB30_USB30="5160" #timeout = 1h34 min
TIMEOUT_LinuxST_Simultaneous_USB30_OTHERS="4320" #timeout = 48 min
TIMEOUT_LinuxST_Simultaneous_HDD_HDD="3120"   #timeout = 1h26 min
TIMEOUT_LinuxST_Simultaneous_OTHERS_HDD="1560" #timeout = 1h12 min
                                                #timeout = 52 min
TIMEOUT_LinuxST_Simultaneous_OTHERS_NFS="2160" #timeout = 26 min
TIMEOUT_LinuxST_Simultaneous_OTHERS_SDCard="3120"
TIMEOUT_LinuxST_Simultaneous_OTHERS_SDLinux="2100" #timeout = 36 min
TIMEOUT_LinuxST_Simultaneous_OTHERS_eMMC="2100" #timeout = 52 min
TIMEOUT_LinuxST_Simultaneous_DVD_OTHERS="3360" #timeout = 35 min
                                                #timeout = 35 min
TIMEOUT_LinuxST_Conflict2Drivers="600"        #timeout = 56 min

TIMEOUT_LinuxST_Benchmark_CPU_3DMark_Ice_Storm="600" #timeout = 10 min
TIMEOUT_LinuxST_Benchmark_CPU_Antutu="480"
TIMEOUT_LinuxST_Benchmark_GPU_3DMark_Ice_Storm="600" #timeout = 10 min
TIMEOUT_LinuxST_Benchmark_GPU_Quadrant_professional="240" #timeout = 8 min
TIMEOUT_LinuxST_Benchmark_IO_Antutu="420"          #timeout = 10 min
TIMEOUT_LinuxST_Benchmark_IO_Quadrant_professional="240" #timeout = 4 min
TIMEOUT_LinuxST_Benchmark_RAM_Antutu="480"         #timeout = 7 min
TIMEOUT_LinuxST_Benchmark_UX_Antutu="480"          #timeout = 4 min

```

- Source: /home/jenkins/overlays/boards/hwreset.config
- Purpose:
  - Define the attendance and device node of Hardware reset.
- Modification:

```

#This file define the attendance of Hardware reset

# HW_RESET="0" #Not support HW reset
HW_RESET="1" #Support HW reset
RVC_HW_RESET_DEVICE="ttyUSB0"

```

- Source: /home/jenkins/tests/LinuxST.../host\_script.sh
- Purpose:
  - Add timeout base for each test case.
- Modification (example for 1 test case):

```
...  
# Define base time out for each test case  
. $FUEGO_SCRIPTS_PATH/timeout_base.sh  
TIMEOUT_BASE=$TIMEOUT_LinuxST_Benchmark_OGLES3Water  
...
```



## 4 Example: Implement a test case

This part shows how to implement a sample test case: transfer 100KB binary data to SD-Card via dd command on R-CarH3 board for Yocto v2.09.00.02.

- **Step 1:** Modify the original FUEGO source code (refer to Part 2 this document for more detail modification)
  - Modify board configuration (please refer to Part 2.3.1 for more details)

- Modified file: /home/jenkins/overlays/boards/renesas.board.salvator-X

```
IPADDR="192.168.1.100"
IPADDR_HOST="192.168.1.1"
LOGIN="root"
FUEGO_HOME="/home/jenkins"
FUEGO_OVERLAYS_PATH="/home/jenkins/overlays"
PASSWORD=""
PLATFORM="renesas-arm"
TRANSPORT="ssh"
ARCHITECTURE="arm"

BOARD.CAP_LIST="RENESAS"

MMC_DEV_LINUX="/dev/mmcblk1p1"
MMC_DIR_LINUX="/mnt/sdcard/"
```

- Modify Main script (please refer to Part 2.3.2 for more details)

- Modified file: /home/jenkins/scripts/LinuxST.sh

```
source $FUEGO_SCRIPTS_PATH/LinuxST_overlays.sh
LinuxST_set_overlay_vars

source $FUEGO_SCRIPTS_PATH/functions.sh
source $FUEGO_SCRIPTS_PATH/LinuxST_functions.sh
source $FUEGO_SCRIPTS_PATH/reports.sh

if $REBOOT; then
    target_reboot
fi

LinuxST_pre_test $TESTDIR

test_compress

if $Rebuild; then
    LinuxST_build
fi

LinuxST_deploy

# Execute Test Script on Target Board
test_run

set_testres_file

# Copy test log from Target Board to Host PC
get_testlog $TESTDIR

# Check Test Log to judge Test Result
LinuxST_fail_check_cases

check_create_logrun
```

- Add new function for Linux ST (mount SD-Card device to Linux File System) (please refer to Part 2.3.4 for more details)

- Modified file: /home/jenkins/scripts/LinuxST\_functions.sh

```
function LinuxST_mount_sdcard() {
    echo "Mount SDCard \n"

    cmd "mkdir -p ${MMC_DIR_LINUX}"
    cmd "umount -f ${MMC_DIR_LINUX} || /bin/true"
    cmd "mount -t vfat ${MMC_DEV_LINUX} ${MMC_DIR_LINUX} || /bin/true"
}
```

- For other scripts (/home/jenkins/scripts/LinuxST\_functions.sh, /home/jenkins/scripts/LinuxST\_overlays.sh), please refer to 2.3.3 for detail modification.
- Add new parser: please refer to Part 2.3.5 to add new parser to FUEGO workspace for detail modification
- **Step 2:** Create host\_script.sh and device\_script.sh
  - Create folder /home/jenkins/tests/LinuxST.Write100k.SDCard
  - In LinuxST.Write100k.SDCard folder, create host\_script.sh and device/device\_script.sh

#### host\_script.sh

```
#!/bin/bash

function test_compress {
    LinuxST_compress_tarball
}

function test_build {
    echo "this test do not need build"
}

function test_deploy {
    put * $FUEGO_HOME/fuego.$TESTDIR/
}

function test_run {
    # Mount SD-Card
    LinuxST_mount_sdcard

    report "cd $FUEGO_HOME/fuego.$TESTDIR; source ./device_script.sh ${MMC_DIR_LINUX} 100K 1"
}

. $FUEGO_ENGINE_PATH/scripts/LinuxST.sh
```

#### device\_script.sh

```
echo "[LINUXST-START] Start Testing"

# set -e
# set -x

if [ $# -ne 3 ]; then
    echo "usage: $(basename $0) PATH BLOCK_SIZE BLOCK_COUNT" >&2
    sleep 0.5
    exit 1
fi

DEV="$1"
BLOCK_SIZE="$2"
BLOCK_COUNT="$3"
IN=/tmp/bin_"${BLOCK_SIZE}"_in
OUT=/tmp/bin_"${BLOCK_SIZE}"_out
DEV_FILE="$DEV"/bin_"${BLOCK_SIZE}"

echo "write test for $DEV (bs=$BLOCK_SIZE count=$BLOCK_COUNT)"
sleep 1

# Check mountpoint
```

```

mountpoint ${DEV} 2>&1
if [ $? -ne 0 ]; then
    echo "Error: ${DESTINATION} Mounting is fail"
    exit 1
fi
echo "${DEV} is mounted"

#IN=""
#OUT=""
cleanup ()
{
    [ -n "$IN" -a -f "#IN" ] &&rm "$IN"
    [ -n "$OUT" -a -f "#OUT" ] &&rm "$OUT"
    #rm -rf /tmp/*
    #rm $DEV_FILE
}
trap cleanup exit
#IN=/tmp/binary
#OUT=$mktemp

echo "Write random data to test file"
busybox dd if=/dev/urandom of="$IN" bs="$BLOCK_SIZE" count="$BLOCK_COUNT"

echo "Write test data to device"
busybox dd if="$IN" of="$DEV_FILE" bs="$BLOCK_SIZE" count="$BLOCK_COUNT"

echo "Read test data from devices"
busybox dd if="$DEV_FILE" of="$OUT" bs="$BLOCK_SIZE" count="$BLOCK_COUNT"

IN_SUM=$(md5sum "$IN" | busybox cut -f 1 -d ' ')
OUT_SUM=$(md5sum "$OUT" | busybox cut -f 1 -d ' ')

IN_SIZE=$(wc -c "$IN" | busybox cut -f 1 -d ' ')
OUT_SIZE=$(wc -c "$OUT" | busybox cut -f 1 -d ' ')

echo "Compare data written to data read"
if [ "$IN_SUM" != "$OUT_SUM" ]; then
    echo "Data read does not match data written"
    echo "Size (bytes):" >&2
    echo "    in: $IN_SIZE" >&2
    echo "    in: $OUT_SIZE" >&2
    echo "SHA 256 Checksums:" >&2
    echo "    in: $IN_SUM" >&2
    echo "    in: $OUT_SUM" >&2
    exit 1
fi

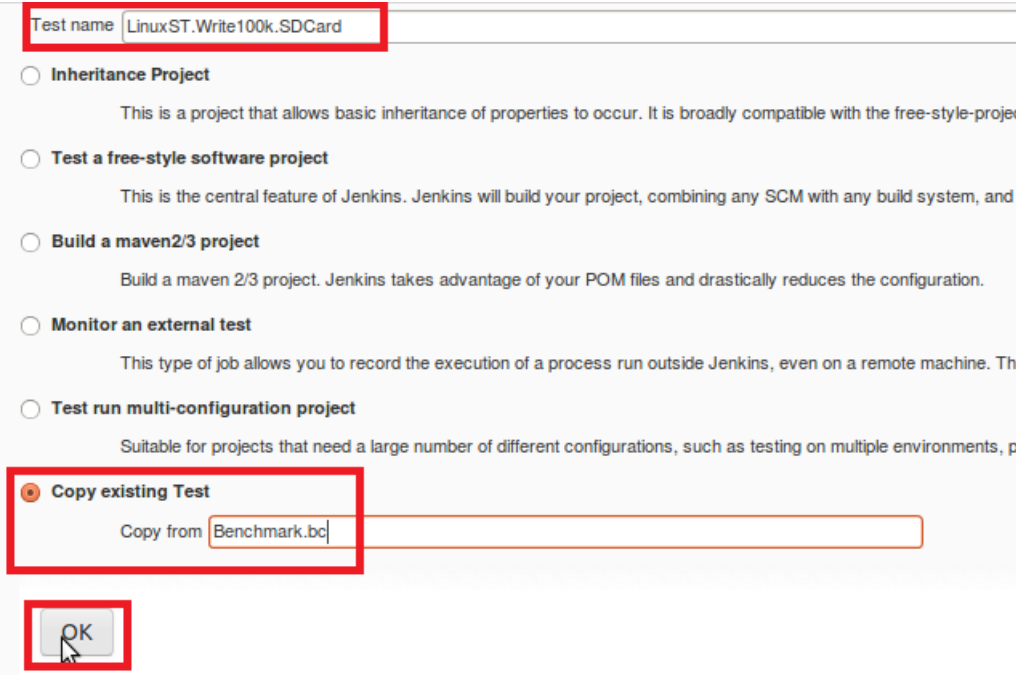
echo "[LINUXST-END] End Testing"

echo "[LINUXST -RESULT-OK]"

```

- **Step 3:** Create test case on Jenkins web interface

Home -> new Test case



Test name

☐ Inheritance Project  
This is a project that allows basic inheritance of properties to occur. It is broadly compatible with the free-style-project

☐ Test a free-style software project  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and

☐ Build a maven2/3 project  
Build a maven 2/3 project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

☐ Monitor an external test  
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. Thi

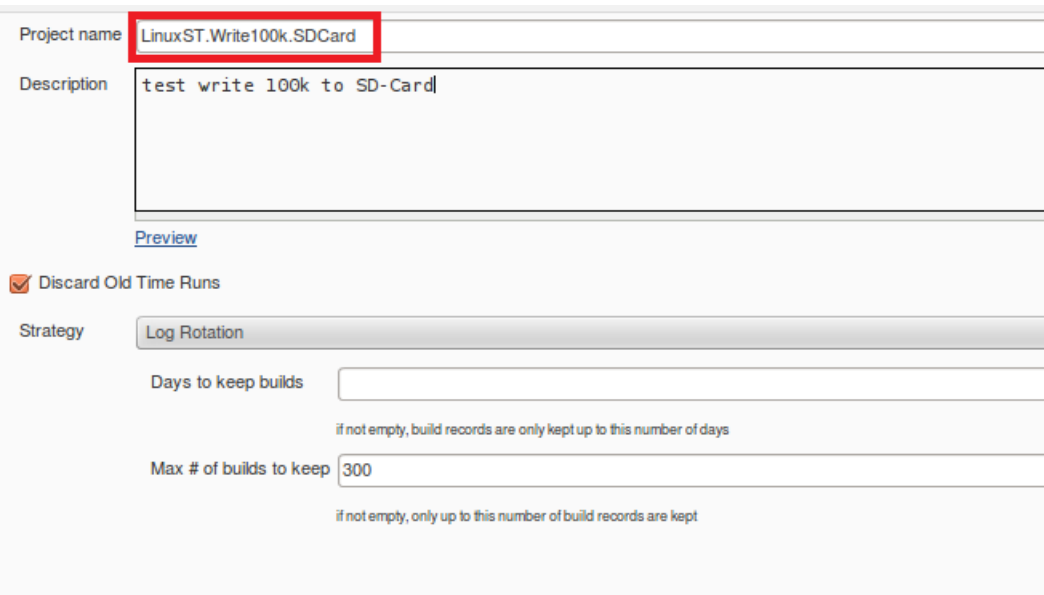
☐ Test run multi-configuration project  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, p

☒ Copy existing Test  
Copy from

Figure 21: Create new test case

Test case configuration:

- Check test case name
- Remove Test Plan option



Project name

Description

[Preview](#)

☒ Discard Old Time Runs

Strategy

Days to keep builds

if not empty, build records are only kept up to this number of days

Max # of builds to keep

if not empty, only up to this number of build records are kept

Figure 22: Test case configuration

Configure test run script:

- Set script of test run as below figure
- Remove all post-test actions
- Save test case

Figure 23: Configure test run script

- **Step 4:** Execute test case
  - Select above created test case (LinuxST.Write.100K.SDCard)
  - Select Run Test Now, select Board and Run Test

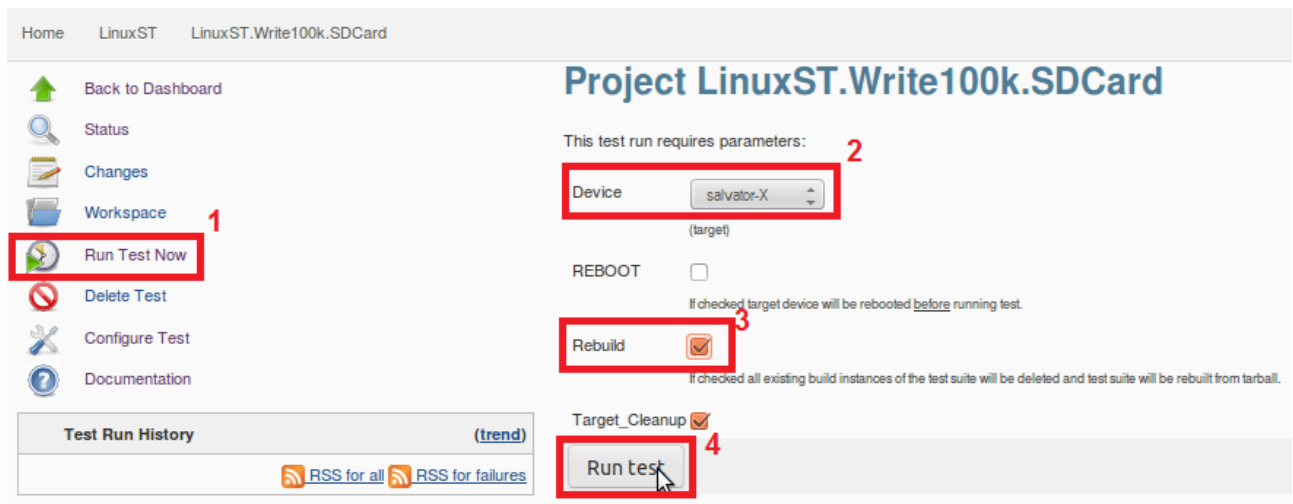


Figure 24: Execute test case

After execution, Test Run History is updated; select Console Output to view detail of test:

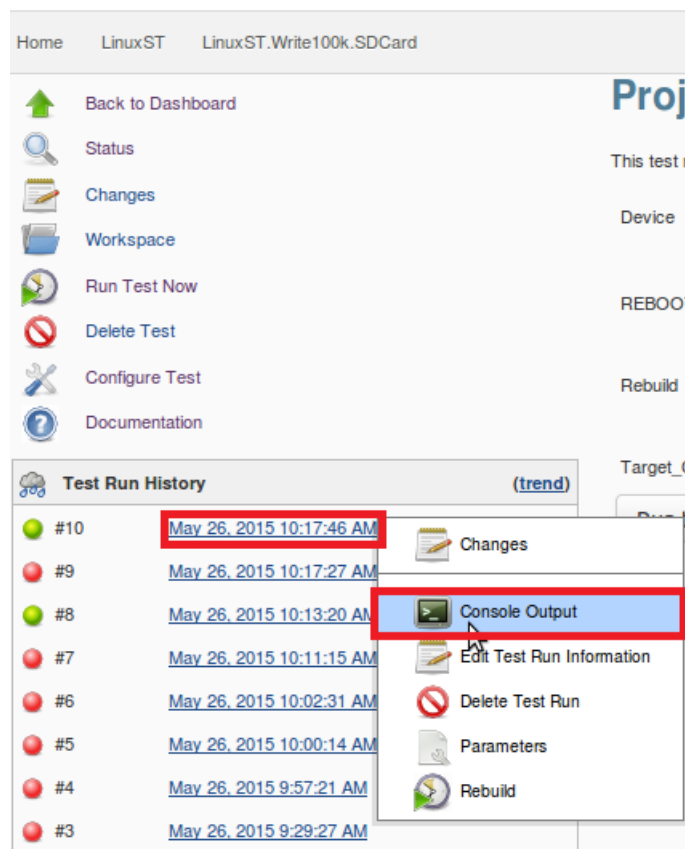


Figure 25: Show console output

**\* Note:**

- Green Ball: test is passed
- Red Ball: test is failed
- While Ball: test is aborted (test case is aborted if connection between Host PC and target board is fail)

Log file:

- At target board: /home/jenkins/ LinuxST.Write100k.SDCard/  
salvator-X.2016-06-06\_11-07-42.110.log
- At Host PC: /home/jenkins/logs/  
LinuxST.Write100k.SDCard/testlogs/salvator-X.2016-06-06\_11-07-42.110.log

```
[LINUXST-START] Start Testing
write test for /mnt/sdcard/ (bs=100K count=1)
/mnt/sdcard/ is a mountpoint
/mnt/sdcard/ is mounted
Write random data to test file
1+0 records in
1+0 records out
102400 bytes (100.0KB) copied, 0.025371 seconds, 3.8MB/s
Write test data to device
1+0 records in
1+0 records out
102400 bytes (100.0KB) copied, 0.002004 seconds, 48.7MB/s
Read test data from devices
1+0 records in
1+0 records out
102400 bytes (100.0KB) copied, 0.000618 seconds, 158.0MB/s
Compare data written to data read
[LINUXST-END] End Testing
[LINUXST-RESULT-OK]
```

## 5 Restriction

- (1) Currently, not support termination of test case when execution time-out is over. (Execution time-out means the limited execution time of one test case)



## 6 Reference

- (1) Website : <http://ltsi.linuxfoundation.org/ltsi-test-project> - LTSI test project from Linux foundation
- (2) Website : <https://bitbucket.org/cogentembedded/fuego-public/> - JTA source code and documents
- (3) 自動テスト環境ガイドライン資料\_rev02.docx – JTA start-up guide from REL on [Redmine](#)
- (4) [ATF\\_Analysis\\_rev01.xlsx](#) – material of JTA investigation from Nomura-san
- (5) Implementation\_Spec\_ATF\_AndroidST\_v1.1.doc – FUEGO implementation specification for AndroidST from RVC
- (6) Website : <https://bitbucket.org/cogentembedded/fuego/> - FUEGO source code and documents
- (7) [jta\\_fuego\\_install\\_memo\\_v1.1.xlsx](#) - Memory note about installing Fuego and JTA on [#83666](#)
- (8) <https://www.docker.com/what-docker#> - Docker's documents and support
- (9) <http://bird.org/fuego/FrontPage> - Fuego Wiki page

## 7 Appendix

### 7.1 Commit change from JTA to Fuego

The below diagram explain briefly about changing in LTSI Test Project (Old JTA -> New JTA -> Fuego)



Currently, RVC is using “FUEGO” for developing Automation Test Framework for Linux System Test.

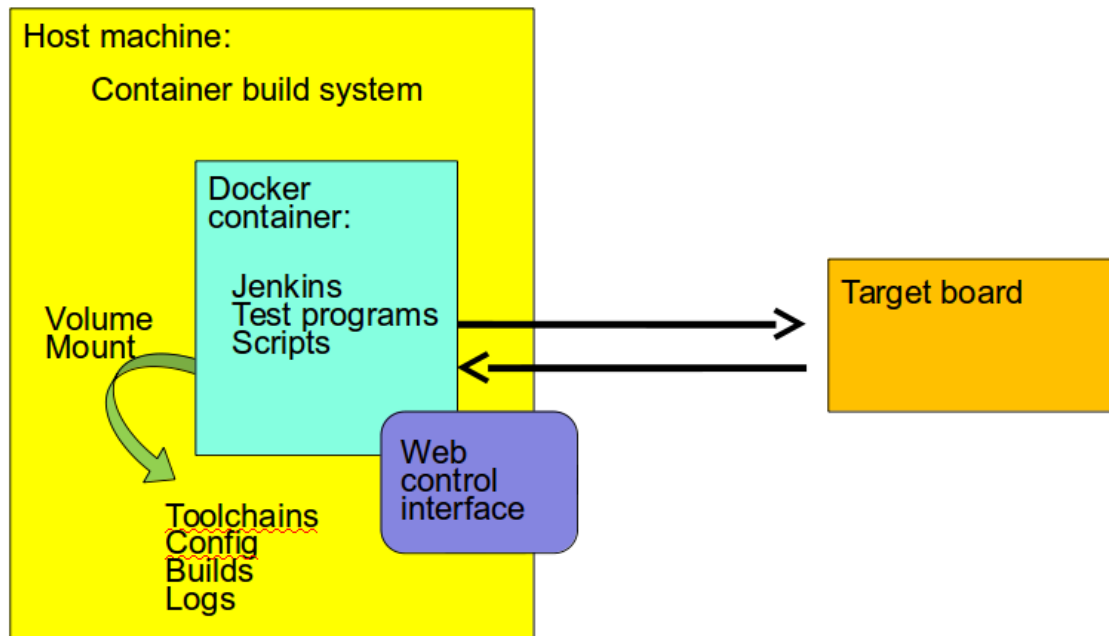
Below is the explanation about available git repositories: <https://bitbucket.org/cogentembedded/>

Repository	Last updated	
fuego	2016-03-25	➤ Rename from jta-public with additional patches
fuego-core	2016-03-25	➤ Rename from jta-core with additional patches
meta-fuego	2016-03-21	➤ Rename from meta-jta with additional patches
jta-core	2016-03-18	➤ Update on top of Jenkins, e.g overlay, sample test plan, test scripts
jta-public	2016-03-16	➤ Scripts to setup JTA environment, support Docker
jta-public-old	2015-12-01	➤ Scripts to setup JTA environment, NOT support Docker
meta-jta	2015-11-30	➤ Yocto recipes to build necessary tools by JTA
jta-tools-public	2014-07-18	➤ Pre-build Yocto SDK (taken from Gen2 Yocto)

*Rename JTA → FUEGO*

Here's a diagram with an overview of Fuego elements:

## Architecture Diagram



## 7.2 Modification to migrate Linux ST from JTA (without Docker) to FUEGO

Below table is used to migrate Linux ST from Old JTA to FUEGO. It is used for person who has familiar with JTA

path&filename				JTA Scripts → Fuego Scripts	Modify/Update Points	
					Original	Correction
/home/jenkins	overlays	base	base-board.jtaclass	Edit the difference	-	Add "ADB" transport
			base-distrib.jtaclass	Edit the difference	-	-
			base-params.jtaclass	Edit the difference	-	-
		boards	hwreset.config	copy	-	-
			renesas.board.salvator-X	copy	JTA_HOME="/home/jenkins"	→ FUEGO_HOME="/home/jenkins" Add: FUEGO_OVERLAYS_PATH="/home/jenkins/overlays"
		distrib	nologger.dist	Edit the difference	-	-
	scripts	ST_device	linux_commons.sh	copy	-	-
			ttyecho	copy	-	-
		app_name.sh		copy	-	-
		LinuxST_common.sh		copy	-	-
		LinuxST.sh		copy	\${JTA_HOME} \${JTA_ENGINE_PATH}/scripts Or \$WORKSPACE/./scripts \${JTA_ENGINE_PATH}/tests Or \$WORKSPACE/./tests \${JTA_ENGINE_PATH}/logs Or \$WORKSPACE/./logs	→ \${FUEGO_HOME} → \${FUEGO_SCRIPTS_PATH} → \${FUEGO_TESTS_PATH} → \${FUEGO_LOGS_PATH}
		LinuxST_functions.sh		copy	\${JTA_ENGINE_PATH}/overlays	→ \${FUEGO_OVERLAYS_PATH}
		LinuxST_overlays.sh		copy	\${JTA_ENGINE_PATH}	→ \${FUEGO_ENGINE_PATH}
		ST_reports.sh		copy	xxx/jta.xxx	→ xxx/fuego.xxx
		LinuxST_fail_token.txt		copy	-	-
		stream_name.sh		copy	-	-
		timeout_base.sh		copy	-	-
		video_functions.sh		copy	-	-
		loggen	LinuxST_loggen.py	copy	-	-
			ST_gentxml.py	copy	-	-
			ST_genxml.py	copy	-	-
			ST_loggen.py	copy	-	-
		parser	LinuxST_parser.py	copy	-	-
			video_parser.py	copy	-	-
	tests	LinuxST.xxxx	host_script.sh	copy	\${JTA_HOME} \${JTA_ENGINE_PATH}/scripts xxx/jta.xxx	→ \${FUEGO_HOME} → \${FUEGO_SCRIPTS_PATH} → xxx/fuego.xxx
			device/device_script.sh	copy	-	-