Industrial Native Real-Time Linux OS Solution

Setup and Tuning Guide

January 2021

Document Number: 634647-1.0

Contents

Tables

Revision History

Date	Revision	Description
August 2020	0.7	Added Coffee Lake and Coffee Lake-R support. Updated script and resource package (RDC# 630356).
January 2021	1.0	Initial release for v1.0.

§

1.0 Introduction

1.1 Intel® Industrial Native Real-Time Linux OS Solution

With Intel® Edge Controls for Industrial (ECI), businesses can control manufacturing like never before. Intel® Edge Controls for Industrial (ECI) is a reference software platform that combines elements of real-time compute, workload consolidation, industrial connectivity, security and functional safety, and software and infrastructure management. ECI provides variant solutions with native OS (without virtualization) or with virtualization. The ECI variance with native OS is referred to as Edge Controls for Industrial-BareMetal (ECI-B).

You may refer to below link for more ECI related information
https://software.intel.com/content/www/us/en/develop/topics/iot/edge-solutions/controls-for-industrial.html

Intel® Edge Controls for Industrial (ECI) is a suite of BitBake* recipes released with Yocto Linux. While Yocto provides customer the full flexibility to create and customize their own Linux distribution, some customers may prefer to develop based on an existing Linux distribution, such as Ubuntu. This document provides customers with a native Real-Time Linux OS solution by porting various key features from ECI-B to Ubuntu, including Preempt RT Linux, Xenomai Linux, and EtherCAT primary stack.

This guide will walk through the porting steps of the Edge Controls for Industrial–BareMetal 1.0 software package onto Ubuntu* 18.04, as well as introduce BKMs on Ubuntu* real-time performance tuning. Customers can choose to use Preempt RT Linux or Xenomai Linux based on their own consideration.

Users of this guide are assumed to be existing ECI-B users, as well as anyone interested in IA real-time solution deployment.

1.2 Software and Hardware Configuration

The main software and hardware configuration are below, but the update will be done in different industrial scenarios.

Table 1. Software and Hardware Configuration

Component	Description
Linux Distribution	Ubuntu 18.04
ECI	ECI v1.0

Component	Description
Hardware Platform	Intel® 8th Generation Core™ U-series processor (Whiskey Lake-U) and Intel® 9th Generation Core™ Refresh processor (Coffee lake-R)

Note: This documentation is taking Intel® Whiskey Lake i7-8665U platform as an example hardware to show the scenario enabling steps.

1.3 The Validated Hardware Platform

This guide is verified on Whiskey Lake-U, Coffee Lake-S, and Coffee Lake-Refresh products.

1.4 Terminology

Table 2. Terminology

Term	Description
Whiskey Lake	Whiskey Lake System on a Chip. An Intel® architecture SoC that integrates the next-generation Intel processor core, graphics, memory controller, and I/O interfaces.
Coffee Lake	Coffee Lake System on a Chip. An Intel® architecture SoC that integrates the next-generation Intel processor core, graphics, memory controller, and I/O interfaces.
Coffee Lake-R	Coffee Lake Refresh System on a Chip. An Intel® architecture SoC that integrates the next-generation Intel processor core, graphics, memory controller, and I/O interfaces.
BIOS	Basic Input Output System
BKM	Best Known Method
ECI	Edge Control Industrial
ECI-B	Edge Control Industrial - BareMetal
LTS	Long Term Support
NIC	Network Interface Card/Controller A network card, network adapter, network interface controller (NIC), network interface card, or LAN adapter is a computer hardware component designed to allow computers to communicate over a computer network.
SMM	System Management Mode
SMI	System Management Interrupt The specific and only interrupt that will cause an x86 machine to enter System Management Mode (SMM).

1.5 Prerequisites

Here are the items required for the software installation:

- ECI-B 1.0 release software package
- Ubuntu* 18.04 boot USB disk

§

2.0 Industrial Native Real-Time OS Solution with Preempt RT Linux

2.1 Linux Host Build Machine Setup

2.1.1 Prepare a Linux Host PC as Host Build Machine

```
$ sudo apt-get install git build-essential kernel-package
fakeroot libncurses5-dev libssl-dev ccache bison flex autogen
automake libtool libmodbus-dev libltdl-dev pkg-config curl
```

A Linux*-based Host PC is needed to build the target packages. Ubuntu 18.04 is preferred and verified. Please make sure the packages in boldface are installed. If not, please, use the apt tool to install them.

2.1.2 Download ECI 1.0 Release

Go to https://registrationcenter.intel.com/regcensec/login.aspx to download the ECI 1.0 PV release. Please contact with your Intel representative for the serial number. The below picture is an example.


```
$ cd ~
$ unzip release-ecs_1.0.zip
$ cd release-ecs_1.0
$ cd -
$ tar -zxvf ecs-release.tar.gz
```

Copy ECI 1.0 PV release downloaded to the Linux Host PC and decompress it.

2.1.3 Create Build Scripts

1. Create an empty folder as your working directory on Linux Host PC and enter this new folder. In this guide the working directory is named “ecs-1.0-ubuntu”.

   ```
   $ mkdir ecs-1.0-ubuntu
   $ cd ecs-1.0-ubuntu
   ```

2. Create three build scripts.

   Create build-all.sh:

   ```
   $ vi build-all.sh
   ```

build-all.sh content is:

```
#!/bin/bash

set -e

DIR=$(dirname "$(realpath -s "$0")")

function help_msg()
{
    printf "Usage:\n"
    printf "  build-all.sh <ecs-release-path> <target>\n"
    printf "  <ecs-release-path>: the decompressed ecs release
package path.\n"
    printf "  <target>: xenomai (default) and preempt\n"
    printf "  For example:\n"
    printf "    build-all.sh /home/release-ecs_1.0/ecs-release
xenomai\n"
}

ECS_REL="${1:-none}"
TARGET="${2:-xenomai}"

if [ ! -d ${ECS_REL} ]; then
    printf "No valid ECS release package path provided! \n"
    help_msg
    exit -1
fi

if [ -f "${DIR}/build-${TARGET}.sh" ]
then
    printf "Start building target ${TARGET} ... \n\n\n"
    source ${DIR}/build-${TARGET}.sh
else
    printf "No valid target provided! \n"
    help_msg
    exit -1
fi
```

Create build-preempt.sh for Preempt-RT build. (You can skip this script if you are using Xenomai Linux)

```
$ vi build-preempt.sh
```

build-preempt.sh content is:

```
#!/bin/bash

set -e

ECS_XENOMAI_META="${ECS_REL}/resources/bare-metal-xenomai"

# Xenomai build parameters
```

```
XENO_META="${ECS_XENOMAI_META}/meta-xenomai-bkc/recipes-
xenomai/xenomai/files"
XENO_VER=3.1
XENO_URL="https://gitlab.denx.de/Xenomai/xenomai/-
/archive/v${XENO_VER}/xenomai-v${XENO_VER}.tar.gz"
XENO_SRC=${DIR}/xenomai
XENO_DEST=${DIR}/xenomai_dest
XENO_RESOURCE=${DIR}/resources/xenomai/xenomai_${XENO_VER}

# Kernel build parameters
KERNEL_VER=4.19.59
KERNEL_TAG=lts-v${KERNEL_VER}-base-190722T171340Z
KERNEL_URL="https://github.com/intel/linux-intel-
lts/archive/${KERNEL_TAG}.tar.gz"
KERNEL_SRC=${DIR}/linux-kernel
KERNEL_RESOURCE=${DIR}/resources/xenomai/linux-
kernel_${KERNEL_VER}
KERNEL_CUSTOM=-ecs1.0

# Ethercat build parameters
ETHERCAT_META="${ECS_XENOMAI_META}/meta-intel-
fieldbus/recipes-ethercat/ighethercat/files"
ETHERCAT_VER="1.5.2"
ETHERCAT_URL="http://www.etherlab.org/download/ethercat/etherc
at-${ETHERCAT_VER}.tar.bz2"
ETHERCAT_SRC=${DIR}/ethercat
ETHERCAT_RESOURCE=${DIR}/resources/xenomai/ethercat_${ETHERCAT
_VER}
ETHERCAT_DEST=${DIR}/ethercat_dest

function download_linux_kernel()
{
    printf "Downloading kernel source from ${KERNEL_URL}\n"
    printf "Kernel tag : ${KERNEL_TAG}\n"

    curl -o linux-kernel.tar.gz -L ${KERNEL_URL}
    rm -rf ${KERNEL_SRC} && mkdir ${KERNEL_SRC}
    tar zxf linux-kernel.tar.gz -C $KERNEL_SRC --strip-
components 1
}

function download_xenomai()
{
    printf "Downloading xenomai source from ${XENO_URL}\n"
    printf "Xenomai tag : ${XENO_VER}\n"

    curl -o xenomai.tar.gz -L ${XENO_URL}
    rm -rf ${XENO_SRC} && mkdir ${XENO_SRC}
    tar zxf xenomai.tar.gz -C ${XENO_SRC} --strip-components 1
}

function download_ethercat()
{
```

```
    printf "Downloading ethercat source from
${ETHERCAT_URL}\n"
    printf "Ethercat tag : ${ETHERCAT_VER}\n"

    curl -o ethercat.tar.bz2 -L ${ETHERCAT_URL}
    rm -rf ${ETHERCAT_SRC} && mkdir ${ETHERCAT_SRC}
    tar jxf ethercat.tar.bz2 -C ${ETHERCAT_SRC} --strip-
components 1
}

function do_patching()
{
    if (($# != 3)); then
        printf "Please provide patch config file, target
source path \n"
        printf "and ECS release resource meta path. \n"
        printf "do_patching ./linux-kernel ./patch
${XENO_META}\n"
        exit -1
    fi

    local src_path=$1
    local patch_cfg=$2
    local meta_path=$3

    pushd ${src_path}

    cat ${patch_cfg} | while read line
    do
        pf=($line)
        if [[ ${pf[0]} == "patch" ]]
        then
            printf "Patching ${pf[1]}\n"
            patch -p1 < ${meta_path}/${pf[1]}
        fi
    done

    popd
}

function build_kernel()
{
    printf "Patching kernel\n"
    do_patching ${KERNEL_SRC} ${KERNEL_RESOURCE}/patch
${XENO_META}

    if [ -f ${KERNEL_RESOURCE}/patches/patch ]; then
        printf "Extra patching\n"
        do_patching ${KERNEL_SRC}
${KERNEL_RESOURCE}/patches/patch ${KERNEL_RESOURCE}/patches
    fi

    printf "Preparing kernel\n"
```

```
    cp ${KERNEL_RESOURCE}/config ${KERNEL_SRC}/.config
    ${XENO_SRC}/scripts/prepare-kernel.sh --arch=x86_64 --
linux=${KERNEL_SRC}

    printf "Building kernel\n"
    pushd ${KERNEL_SRC}
    make -j `getconf _NPROCESSORS_ONLN` bindeb-pkg
LOCALVERSION=${KERNEL_CUSTOM}
    popd
}

function build_xenomai()
{
    printf "Patching xenomai\n"
    do_patching ${XENO_SRC} ${XENO_RESOURCE}/patch
${XENO_META}

    printf "Building xenomai\n"
    rm -rf ${XENO_DEST} && mkdir ${XENO_DEST}
    pushd ${XENO_SRC}

    ./scripts/bootstrap
    ./configure --prefix=/usr/xenomai  --with-core=cobalt  --
enable-smp  --enable-pshared
    make -j `getconf _NPROCESSORS_ONLN`
    make DESTDIR=${XENO_DEST} install

    popd
}

function build_ethercat()
{
    printf "Patching ethercat\n"
    do_patching ${ETHERCAT_SRC} ${ETHERCAT_RESOURCE}/patch
${ETHERCAT_META}

    printf "Building ethercat\n"
    rm -rf ${ETHERCAT_DEST} && mkdir ${ETHERCAT_DEST}
    pushd ${ETHERCAT_SRC}
    local xenomai_dir=${XENO_DEST}/usr/xenomai

# workaround to make ethercat build on the previous built
xenomai
#    cp ${xenomai_dir}/bin/xeno-config ${DIR}/xeno-config
#    sed -i
"s#prefix=\"/usr/xenomai\"#prefix=\"${xenomai_dir}\"#g"
${DIR}/xeno-config

    sudo cp -r ${xenomai_dir} /usr/
    chmod +x bootstrap && ./bootstrap
    ./configure --enable-sii-assign \
                --disable-8139too \
                --enable-igb \
```

```
                --libdir=/usr/xenomai/lib \
                --with-xenomai-dir=/usr/xenomai \
                --with-xenomai-config=/usr/xenomai/bin/xeno-
config \
                --with-linux-dir=${KERNEL_SRC} \
                --with-devices=8 \
                --enable-rtdm \
                --disable-eoe \
                --enable-hrtimer \
                --enable-cycles
    make all modules
    make DESTDIR=${ETHERCAT_DEST} install
    make INSTALL_MOD_PATH=${ETHERCAT_DEST} modules_install
    popd
    sudo rm -rf /usr/xenomai
}

function pack_xenomai_to_deb()
{
    printf "Packaging xenomai to deb\n"

    mkdir ${XENO_DEST}/DEBIAN
    cp ${XENO_RESOURCE}/control ${XENO_DEST}/DEBIAN/
    dpkg -b ${XENO_DEST} xenomai_${XENO_VER}_amd64.deb
}

function pack_ethercat_to_deb()
{
    printf "Packaging ethercat to deb\n"

    # workaround to delete modules info files to avoid install
conflict
    rm -f
${ETHERCAT_DEST}/lib/modules/${KERNEL_VER}*/modules.*
    mkdir ${ETHERCAT_DEST}/DEBIAN
    cp ${ETHERCAT_RESOURCE}/control ${ETHERCAT_DEST}/DEBIAN/
    dpkg -b ${ETHERCAT_DEST}
ethercat_${ETHERCAT_VER}_amd64.deb
}


if [ ! -d ${XENO_META} ]; then
    printf "Path ${XENO_META} is not existed!\n"
    printf "No valid ECS release package path provided! \n"
    exit -1
fi

download_linux_kernel
download_xenomai
download_ethercat
build_kernel
build_xenomai
build_ethercat
```

```
pack_xenomai_to_deb
pack_ethercat_to_deb
```

Create build-xenomai.sh for the Xenomai build. (You can skip this script if you are going to using Preempt RT Linux.)

```
$ vi build-xenomai.sh

build-xenomai.sh content is:
#!/bin/bash

set -e

ECS_XENOMAI_META="${ECS_REL}/resources/bare-metal-xenomai"

# Xenomai build parameters
XENO_META="${ECS_XENOMAI_META}/meta-xenomai-bkc/recipes-
xenomai/xenomai/files"
XENO_VER=3.1
XENO_URL="https://gitlab.denx.de/Xenomai/xenomai/-
/archive/v${XENO_VER}/xenomai-v${XENO_VER}.tar.gz"
XENO_SRC=${DIR}/xenomai
XENO_DEST=${DIR}/xenomai_dest
XENO_RESOURCE=${DIR}/resources/xenomai/xenomai_${XENO_VER}

# Kernel build parameters
KERNEL_VER=4.19.59
KERNEL_TAG=lts-v${KERNEL_VER}-base-190722T171340Z
KERNEL_URL="https://github.com/intel/linux-intel-
lts/archive/${KERNEL_TAG}.tar.gz"
KERNEL_SRC=${DIR}/linux-kernel
KERNEL_RESOURCE=${DIR}/resources/xenomai/linux-
kernel_${KERNEL_VER}
KERNEL_CUSTOM=-ecs1.0

# Ethercat build parameters
ETHERCAT_META="${ECS_XENOMAI_META}/meta-intel-
fieldbus/recipes-ethercat/ighethercat/files"
ETHERCAT_VER="1.5.2"
ETHERCAT_URL="http://www.etherlab.org/download/ethercat/etherc
at-${ETHERCAT_VER}.tar.bz2"
ETHERCAT_SRC=${DIR}/ethercat
ETHERCAT_RESOURCE=${DIR}/resources/xenomai/ethercat_${ETHERCAT
_VER}
ETHERCAT_DEST=${DIR}/ethercat_dest

function download_linux_kernel()
{
    printf "Downloading kernel source from ${KERNEL_URL}\n"
    printf "Kernel tag : ${KERNEL_TAG}\n"

    curl -o linux-kernel.tar.gz -L ${KERNEL_URL}
```

```
    rm -rf ${KERNEL_SRC} && mkdir ${KERNEL_SRC}
    tar zxf linux-kernel.tar.gz -C $KERNEL_SRC --strip-
components 1
}

function download_xenomai()
{
    printf "Downloading xenomai source from ${XENO_URL}\n"
    printf "Xenomai tag : ${XENO_VER}\n"

    curl -o xenomai.tar.gz -L ${XENO_URL}
    rm -rf ${XENO_SRC} && mkdir ${XENO_SRC}
    tar zxf xenomai.tar.gz -C ${XENO_SRC} --strip-components 1
}

function download_ethercat()
{
    printf "Downloading ethercat source from
${ETHERCAT_URL}\n"
    printf "Ethercat tag : ${ETHERCAT_VER}\n"

    curl -o ethercat.tar.bz2 -L ${ETHERCAT_URL}
    rm -rf ${ETHERCAT_SRC} && mkdir ${ETHERCAT_SRC}
    tar jxf ethercat.tar.bz2 -C ${ETHERCAT_SRC} --strip-
components 1
}
function do_patching()
{
    if (($# != 3)); then
        printf "Please provide patch config file, target
source path \n"
        printf "and ECS release resource meta path. \n"
        printf "do_patching ./linux-kernel ./patch
${XENO_META}\n"
        exit -1
    fi

    local src_path=$1
    local patch_cfg=$2
    local meta_path=$3

    pushd ${src_path}

    cat ${patch_cfg} | while read line
    do
        pf=($line)
        if [[ ${pf[0]} == "patch" ]]
        then
            printf "Patching ${pf[1]}\n"
            patch -p1 < ${meta_path}/${pf[1]}
        fi
    done
```

```
    popd
}

function build_kernel()
{
    printf "Patching kernel\n"
    do_patching ${KERNEL_SRC} ${KERNEL_RESOURCE}/patch
${XENO_META}

    if [ -f ${KERNEL_RESOURCE}/patches/patch ]; then
        printf "Extra patching\n"
        do_patching ${KERNEL_SRC}
${KERNEL_RESOURCE}/patches/patch ${KERNEL_RESOURCE}/patches
    fi

    printf "Preparing kernel\n"
    cp ${KERNEL_RESOURCE}/config ${KERNEL_SRC}/.config
    ${XENO_SRC}/scripts/prepare-kernel.sh --arch=x86_64 --
linux=${KERNEL_SRC}

    printf "Building kernel\n"
    pushd ${KERNEL_SRC}
    make -j `getconf _NPROCESSORS_ONLN` bindeb-pkg
LOCALVERSION=${KERNEL_CUSTOM}
    popd
}

function build_xenomai()
{
    printf "Patching xenomai\n"
    do_patching ${XENO_SRC} ${XENO_RESOURCE}/patch
${XENO_META}

    printf "Building xenomai\n"
    rm -rf ${XENO_DEST} && mkdir ${XENO_DEST}
    pushd ${XENO_SRC}

    ./scripts/bootstrap
    ./configure --prefix=/usr/xenomai  --with-core=cobalt  --
enable-smp  --enable-pshared
    make -j `getconf _NPROCESSORS_ONLN`
    make DESTDIR=${XENO_DEST} install

    popd
}

function build_ethercat()
{
    printf "Patching ethercat\n"
    do_patching ${ETHERCAT_SRC} ${ETHERCAT_RESOURCE}/patch
${ETHERCAT_META}

    printf "Building ethercat\n"
```

```
    rm -rf ${ETHERCAT_DEST} && mkdir ${ETHERCAT_DEST}
    pushd ${ETHERCAT_SRC}
    local xenomai_dir=${XENO_DEST}/usr/xenomai
# workaround to make ethercat build on the previous built
xenomai
#   cp ${xenomai_dir}/bin/xeno-config ${DIR}/xeno-config
#   sed -i
"s#prefix=\"/usr/xenomai\"#prefix=\"${xenomai_dir}\"#g"
${DIR}/xeno-config

    sudo cp -r ${xenomai_dir} /usr/
    chmod +x bootstrap && ./bootstrap
    ./configure --enable-sii-assign \
                --disable-8139too \
                --enable-igb \
                --libdir=/usr/xenomai/lib \
                --with-xenomai-dir=/usr/xenomai \
                --with-xenomai-config=/usr/xenomai/bin/xeno-
config \
                --with-linux-dir=${KERNEL_SRC} \
                --with-devices=8 \
                --enable-rtdm \
                --disable-eoe \
                --enable-hrtimer \
                --enable-cycles
    make all modules
    make DESTDIR=${ETHERCAT_DEST} install
    make INSTALL_MOD_PATH=${ETHERCAT_DEST} modules_install
    popd
    sudo rm -rf /usr/xenomai
}

function pack_xenomai_to_deb()
{
    printf "Packaging xenomai to deb\n"

    mkdir ${XENO_DEST}/DEBIAN
    cp ${XENO_RESOURCE}/control ${XENO_DEST}/DEBIAN/
    dpkg -b ${XENO_DEST} xenomai_${XENO_VER}_amd64.deb
}

function pack_ethercat_to_deb()
{
    printf "Packaging ethercat to deb\n"

    # workaround to delete modules info files to avoid install
conflict
    rm -f
${ETHERCAT_DEST}/lib/modules/${KERNEL_VER}*/modules.*
    mkdir ${ETHERCAT_DEST}/DEBIAN
    cp ${ETHERCAT_RESOURCE}/control ${ETHERCAT_DEST}/DEBIAN/
    dpkg -b ${ETHERCAT_DEST}
ethercat_${ETHERCAT_VER}_amd64.deb
```

```
}


if [ ! -d ${XENO_META} ]; then
    printf "Path ${XENO_META} is not existed!\n"
    printf "No valid ECS release package path provided! \n"
    exit -1
fi

download_linux_kernel
download_xenomai
download_ethercat
build_kernel
build_xenomai
build_ethercat
pack_xenomai_to_deb
pack_ethercat_to_deb
```

3. Add execute permissions to the build scripts created:

```
$ cd <your path to>/ecs-1.0-ubuntu
$ chmod +x *.sh
```

2.1.4 Create Configuration Files

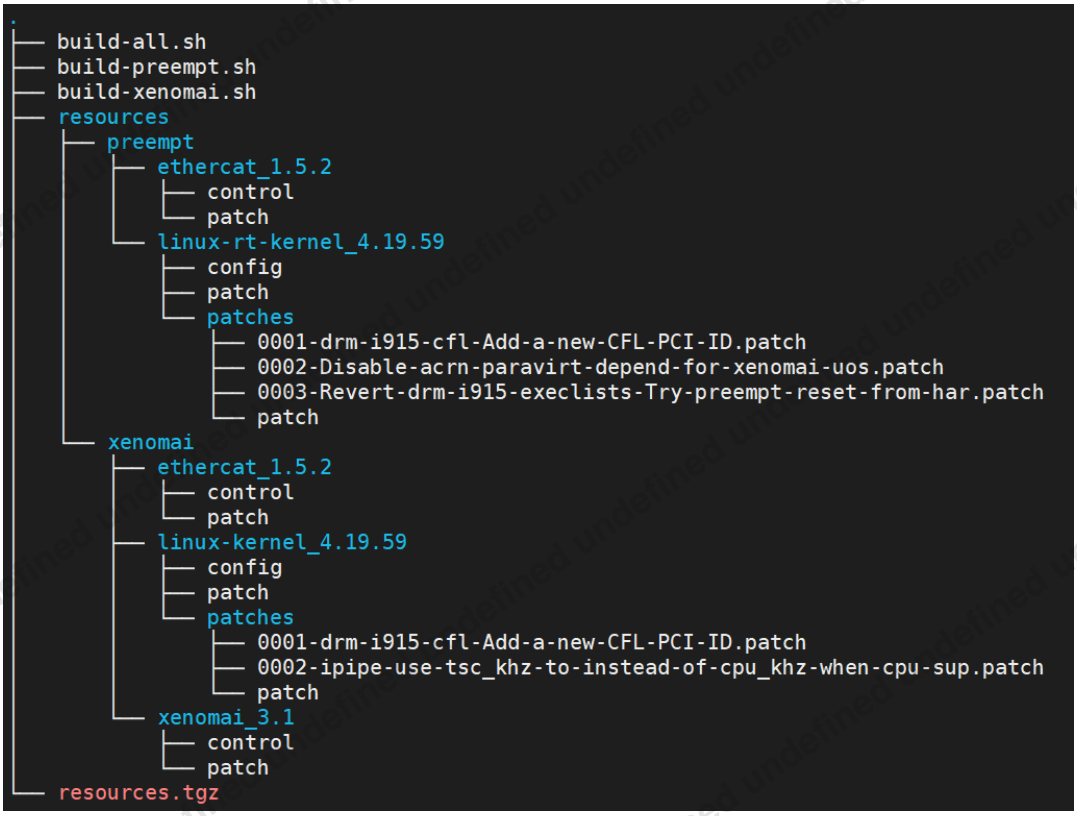
1. Download the 'resources.tgz' file to the Linux* Host PC. Copy it to the working directory "ecs-1.0-ubuntu" and unzip it.

Note: Download the 'resources.tgz' file from Intel RDC #630356.

```
$ cd <your path to>/ecs-1.0-ubuntu
$ cp <your path to>/resource.tgz .
$ tar -zxvf resources.tgz
```

2. Verify the directory structure as shown in the figure below.

```
$ tree .
```



2.2 Install Ubuntu on Target Hardware

Install Ubuntu* OS on the target board. Ubuntu 18.04 is preferred and verified

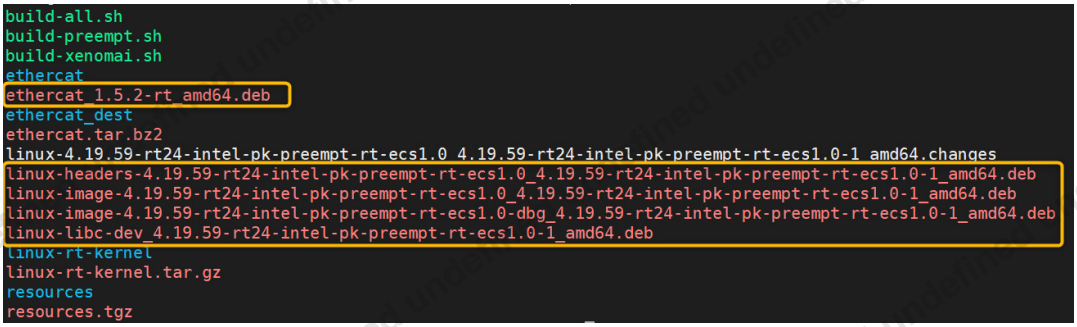
2.3 Build Debian* Packages for Preempt RT Linux

1. On the Linux host build machine, navigate to the working directory “ecs-1.0-ubuntu” and run the build script to build preempt-rt Linux-based Debian* packages. You need to make sure you have completed Section 2.2 and know the absolute path to directory “ecs-release_1.0”

```
$ cd <your path to>/ecs-1.0-ubuntu
$ chmod +x *.sh
$ ./build-all.sh <absolute path>/release-ecs_1.0/ecs-release/
preempt
```

Note: Make sure to use “absolute path” to “release-ecs_1.0”.

2. If build is successful, the Debian packages for Preempt-RT Linux will be created in the folder.



2.4 Preempt RT Linux Installation

1. Copy the Debian packages for Preempt-RT Linux built in previous step to the target board.

2. Install Preempt-RT Linux Debian packages.

```
$ sudo dpkg -i [path]/linux-image-4.19.59-rt24-intel-pk-
preempt-rt-ecs1.0_4.19.59-rt24-intel-pk-preempt-rt-ecs1.0-
1_amd64.deb
$ sudo dpkg -i [path]/linux-headers-4.19.59-rt24-intel-pk-
preempt-rt-ecs1.0_4.19.59-rt24-intel-pk-preempt-rt-ecs1.0-
1_amd64.deb
$ sudo dpkg -i [path]/linux-libc-dev_4.19.59-rt24-intel-pk-
preempt-rt-ecs1.0-1_amd64.deb
$ sudo dpkg -i [path]/ethercat_1.5.2-rt_amd64.deb
```

3. Edit grub to leave time to select new kernel and change the kernel cmdline.

```
$ sudo gedit /etc/default/grub
```

Changes are:

```
#GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=10

GRUB_CMDLINE_LINUX_DEFAULT="quiet splash nosmap
console=ttyS0,115200 console=tty0 clocksource=tsc tsc=reliable
nmi_watchdog=0 nosoftlockup idle=poll noht audit=0
irqaffinity=0 isolcpus=1-3 rcu_nocbs=1-3 intel_pstate=disable
intel.max_cstate=0 intel_idle.max_cstate=0
processor.max_cstate=0 processor_idle.max_cstate=0
i915.enable_dc=0 i915.disable_power_well=0 noefi"
```

NOTE:

1. Usually, the implementation for EFI variable read/write is in SMM, which means it will trigger SMI and result in large jitter. “noefi” will disable software access to EFI variable.
2. User can change isolcpus and rcu_nocbs arguments per needs.

4. Update grub and reboot.

```
$ sudo update-grub
```

```
$ sudo reboot
```

5. After reboot into Grub menu, select “Advanced options for Ubuntu” → “Ubuntu, with Linux 4.19.59-rt24-intel-pk-preempt-rt-ecs1.0” in Grub menu, and then boot.

2.5 Real-Time Performance Test

In this example, we test the real-time performance when system is in idle status and when system is under extra high workload. We use the tool cyclictest from package rt-tests as benchmark tool. The workload is designed to include CPU/GPU/IO/Disk/Scheduler.

To get more accurate result, we suggest to run the test for at least 7x24 hours.

2.5.1 Install Test Packages

Install common test packages.

```
$ sudo apt-get install rt-tests
$ sudo apt-get install stress
$ sudo apt-get install glmark2
$ sudo apt-get install htop
$ sudo apt-get install intel-gpu-tools
```

2.5.2 Perform Real-Time Test

1. Test while in idle status.

   ```
   $ sudo taskset -c 3 cyclictest -smp 99 -a 3 -n -i 250 -D 1d -H
   100 --histfile=hist.log
   ```

   Bind “cyclictest” to core 3, use nanosleep and 250us cycle to do stress test for one day.

2. Test under extra high CPU/GPU/IO/Disk/Scheduler workload.

   ```
   $ sudo taskset -c 3 cyclictest -smp 99 -a 3 -n -i 250 -D 1d -H
   100 --histfile=hist.log
   $ sudo taskset -c 1 glmark2 --run-forever
   $ stress -q -i 4 -q -c 4 -q -d 4 --hdd-bytes 20M -q -m 4 --vm-
   bytes 10M
   $ while true; do sudo taskset -c 1 hackbench -s 512 -l 200 -g
   20 -f 50 -P; done
   ```

2.5.3 Data Check

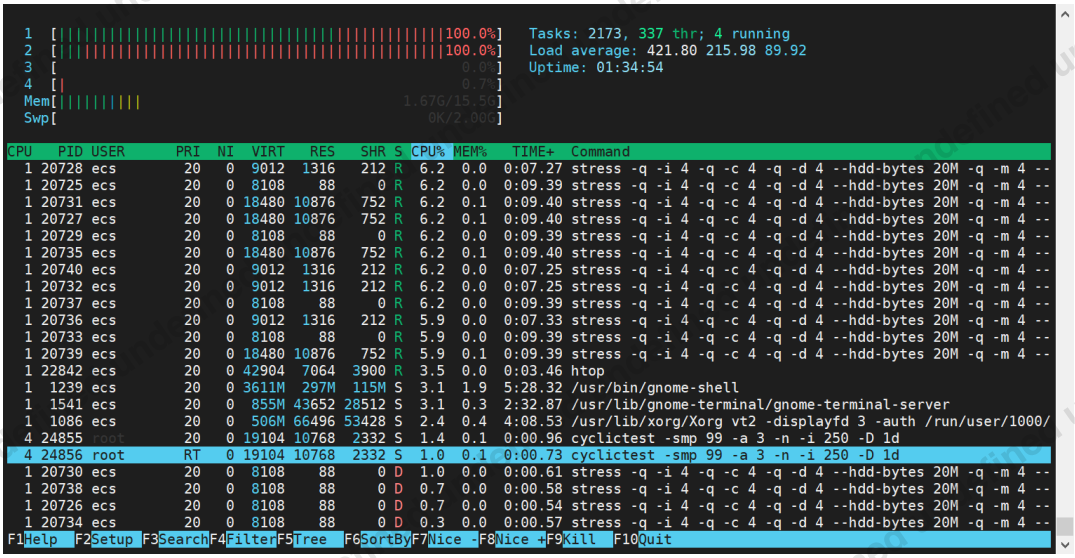
1. Check the jitter reported from “cyclictest”. Here is an example. “Max: 12” means worst jitter is 12us for around 4-minutes test. Check this data after a one-day stress test.

```
ecs@ecs-Default-string:~$ sudo taskset -c 3 cyclictest -smp 99 -a 3 -n -i 250 -D 1d
[sudo] password for ecs:
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 4.22 4.37 6.42 1/629 12505

T: 0 (12504) P:99 I:250 C: 991997 Min:      2 Act:    3 Avg:    3 Max:      12
```

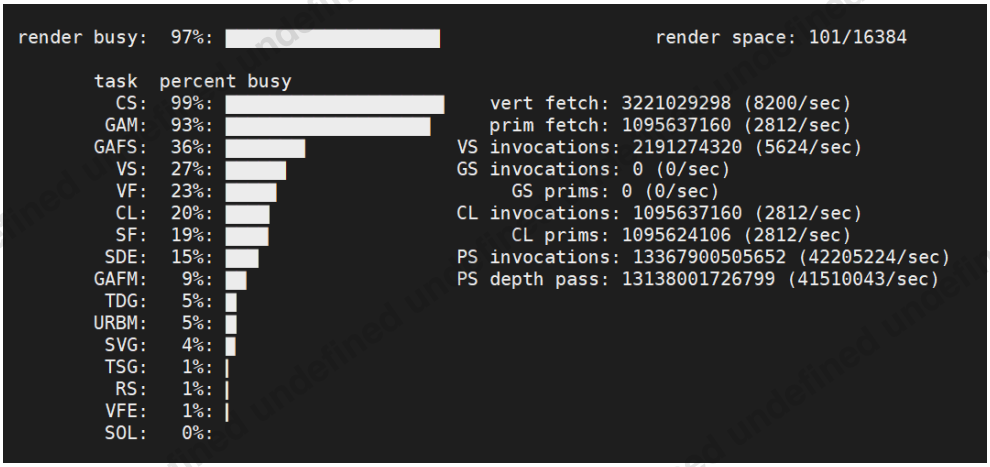
2. Check CPU workload status:

```
$ htop
```



3. Check GPU workload status:

```
$ sudo intel_gpu_top
```



2.5.4 EtherCAT Test

Here is an example on how to enable EtherCAT for testing.

```
$ sudo depmod
$ sudo mkdir /etc/sysconfig
$ sudo cp /opt/etherlab/etc/sysconfig/ethercat
/etc/sysconfig/ethercat
$ sudo gedit /etc/sysconfig/ethercat
Set MASTER0_DEVICE, DEVICE_MODULES(igb) and REBIND_NICS
accordingly per NIC information
$ sudo /opt/etherlab/etc/init.d/ethercat stop
$ sudo /opt/etherlab/etc/init.d/ethercat start
$ sudo /opt/etherlab/bin/ethercat slaves
```

Then, the user can use their EtherCAT APP to test with EtherCAT devices.

§

3.0 Industrial Native Real-Time OS Solution with Xenomai Linux

3.1 Linux Host Build Machine Setup

Please Refer to Section 2.1 for Linux host build machine setup.

3.2 Install Ubuntu on Target Hardware

Install Ubuntu* OS on the target board. Ubuntu 18.04 is preferred and verified

3.3 Build Debian* Packages for Xenomai Linux

1. On the Linux host build machine, navigate to the working directory “ecs-1.0-ubuntu” and run the build script to build Xenomai Linux-based Debian* packages. You need to make sure you have completed Section 2.2 and know the absolute path to directory “ecs-release_1.0”

```
$ cd <your path to>/ecs-1.0-ubuntu
$ chmod +x *.sh
$ ./build-all.sh <absolute path>/release-ecs_1.0/ecs-release/
xenomai
```

Note: Make sure to use “absolute path” to “release-ecs_1.0”.

Note: EtherCAT stack build for Xenomai will ask for “sudo” permission to provide temporary Xenomai environment for EtherCAT build.

2. If build is successful, the Debian packages for Xenomai Linux will be created in the folder.

3.4 Xenomai Linux Installation

1. Copy the Xenomai Linux Debian packages built in the previous step to target board.

   ```
   $ sudo gedit /etc/default/grub
   ```

2. Install Xenomai Linux Debian packages.

   ```
   $ sudo dpkg -i [path]/linux-image-4.19.59-intel-pk-standard-
   ecs1.0_4.19.59-intel-pk-standard-ecs1.0-1_amd64.deb
   $ sudo dpkg -i [path]/linux-headers-4.19.59-intel-pk-standard-
   ecs1.0_4.19.59-intel-pk-standard-ecs1.0-1_amd64.deb
   $ sudo dpkg -i [path]/linux-libc-dev_4.19.59-intel-pk-
   standard-ecs1.0-1_amd64.deb
   $ sudo dpkg -i [path]/ethercat_1.5.2_amd64.deb
   $ sudo dpkg -i [path]/xenomai_3.1_amd64.deb
   ```

3. Edit grub to leave time to select new kernel and change the kernel cmdline.

   Changes are:

   ```
   #GRUB_TIMEOUT_STYLE=hidden
   GRUB_TIMEOUT=10

   GRUB_CMDLINE_LINUX_DEFAULT="quiet splash
   xenomai.allowed_group=1234 xenomai.sysheap_size=256
   xenomai.state=enabled xenomai.smi=detect xenomai.smi_mask=1
   nosmap console=ttyS0,115200 console=tty0 clocksource=tsc
   tsc=reliable nmi_watchdog=0 nosoftlockup idle=poll noht
   audit=0 irqaffinity=0 isolcpus=1-3 rcu_nocbs=1-3
   intel_pstate=disable intel.max_cstate=0
   intel_idle.max_cstate=0 processor.max_cstate=0
   ```

```
processor_idle.max_cstate=0 i915.enable_dc=0
i915.disable_power_well=0 noefi”
```

Note: Please refer to “noefi”, “isolcpus” and “rcu_nocbs” descriptions in Section 2.4.

4. Update grub and reboot.

```
$ sudo update-grub
$ sudo reboot
```

5. After reboot, select “Advanced options for Ubuntu” → “Ubuntu, with Linux 4.19.59-intel-pk-standard-ecs1.0” in the grub menu, and then boot.

3.5 Real-Time Performance Test

In this example, we test the real-time performance when system is in idle status and when system is under extra high workload. We use the tool latency from Xenomai package as benchmark tool. The workload is designed to include CPU/GPU/IO/Disk/Scheduler.

To get more accurate result, we suggest to run the test for at least 7x24 hours.

3.5.1 Install Test Packages

Install common test packages.

```
$ sudo apt-get install rt-tests
$ sudo apt-get install stress
$ sudo apt-get install glmark2
$ sudo apt-get install htop
$ sudo apt-get install intel-gpu-tools
```

3.5.2 Perform Real-Time Test

1. Test while in idle status.

   Bind “latency” to core 3, use 250us cycle to do stress test for one day.

```
$ sudo taskset -c 3 /usr/xenomai/bin/latency -c 3 -T 86400 -p
250 -s -g hist.txt
```

2. Test under extra high CPU/GPU/IO/Disk/Scheduler workload.

```
$ sudo taskset -c 3 /usr/xenomai/bin/latency -c 3 -T 86400 -p
250 -s -g hist.txt
$ sudo taskset -c 1 glmark2 --run-forever
$ stress -q -i 4 -q -c 4 -q -d 4 --hdd-bytes 20M -q -m 4 --vm-
bytes 10M
```

```
$ while true; do sudo taskset -c 1 hackbench -s 512 -l 200 -g
20 -f 50 -P; done
```

3.5.3 Data Check

Check the jitter reported from “latency”. Here is an example. “--lat worst 1.100” means worst jitter is 1.1us for around 4-seconds test. Check this data after a one-day stress test.

```
ecs@ecs-Default-string:~$ sudo taskset -c 3 /usr/xenomai/bin/latency -c 3 -T 86400 -p 250 -s -g hist.txt
[sudo] password for ecs:
== Sampling period: 250 us
== Test mode: periodic user-mode task
== All results in microseconds
warming up...
RTT|  00:00:01  (periodic user-mode task, 250 us period, priority 99)
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst
RTD|      0.077|      0.275|      0.570|       0|     0|      0.077|      0.570
RTD|      0.076|      0.272|      0.990|       0|     0|      0.076|      0.990
RTD|      0.082|      0.276|      0.641|       0|     0|      0.076|      0.990
RTD|      0.075|      0.275|      1.100|       0|     0|      0.075|      1.100
```

You can refer to Section 3.4.3 about the usage of command “htop” to check CPU workload and command “intel_gpu_top” to check GPU workload.

3.5.4 EtherCAT Test

Note: For EtherCAT under Xenomai Linux, please double-check the “clockfreq” below.

```
$ dmesg | grep tsc
[ 0.000000] tsc: Detected 2100.000 MHz processor
[ 0.002890] tsc: Detected 2112.000 MHz TSC, current
tsc:59868787892
```

If it shows offset between processor frequency and TSC frequency on your machine like the example, please edit “GRUB_CMDLINE” in “/etc/default/grub” to add one more argument, “xenomai.clockfreq=[TSC Frequency]“, and then run “update-grub”, and reboot machine.

In this example, add “xenomai.clockfreq=2112000000" into kernel cmdline.

Then please refer to Section 3.4.4 for the introduction of EtherCAT test.

§

4.0 Real-Time Performance Tuning Tips

4.1 BIOS Setting BKM

Please log in to the Intel RDC website and download doc #612900: Whiskey Lake-U Real-Time Performance Tuning Guidance.

4.2 Ubuntu Tips

Disable “blank screen” in Ubuntu Settings/Power. Set it to “Never”.

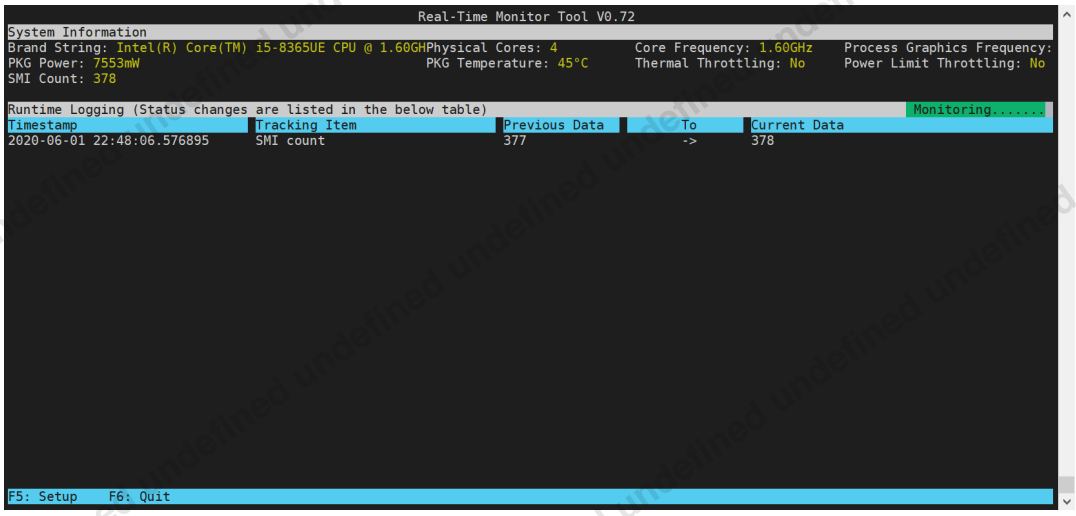
4.3 Real-Time Performance Tuning Suite

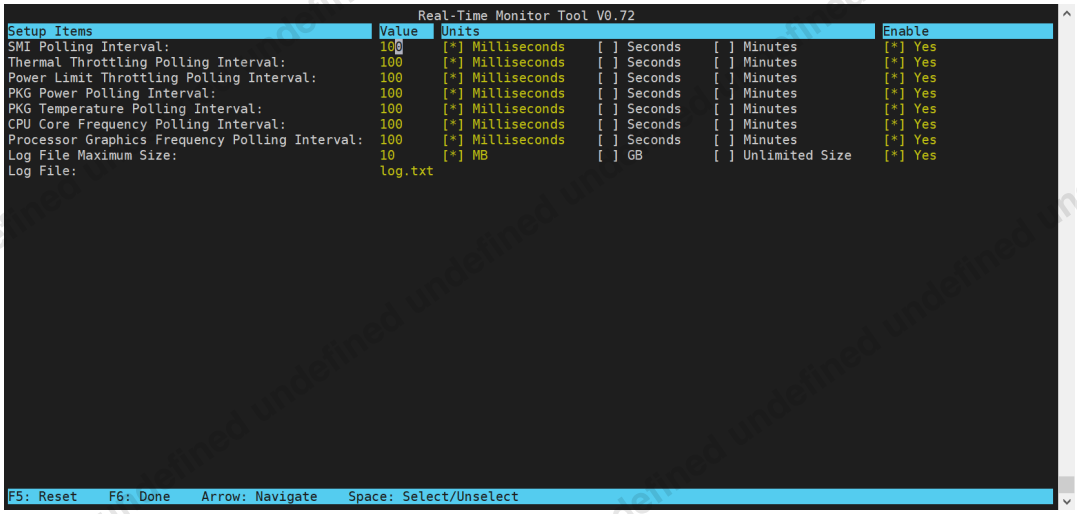
4.3.1 Download Real-Time Performance Tuning Suite

Go to Intel® Validation Internet Portal to download “Real-Time Performance Tuning Suite 1.3”. The direct URL is listed as below.
https://platformsw.intel.com/ddrivers.aspx?kitnumber=1000581

4.3.2 Brief Introduction of Real-Time Performance Tuning Suite

“Intel® Real-Time Monitor Tool (Linux)” is a Linux tool running in Ubuntu to monitor real-time related events or indicators like SMI, CPU/GPU frequency change, power limitation throttling, thermal throttling, package power, and others that may impact real-time performance. If such events happened, the events will be logged into file on the disk and showed in the UI. In this case, no events happening is preferred.





- "Intel® Real-Time Configuration Check Tool" is a tool to check real-time related BIOS settings and give some recommends. It has two versions. One is BIOS version running in UEFI shell, and another is the Linux version running in the Linux shell. Some registers will be locked during silicon boot. So, the BIOS version has more permissions than Linux to dump settings. However, some settings may be overridden by the OS and Linux kernel. The Linux version of the tool can help to find such a case.

CPU Features		
Feature	Status	Recommendation
DRAM content to PRM When CPU in C6	Locked	Disabled
Hyper-Threading Technology	Disabled	Disabled

Power & Performance		
Feature	Status	Recommendation
Intel (R) SpeedStep Technology	Disabled	Disabled
Intel (R) Speed Shift Technology	Enabled	Disabled
CPU Power Management: C-State	Disabled	Disabled
GT Render Standby (RC6)	Disabled	Disabled

§