**MICRO CONTROLLER UNIT (MCU) - TESTING**

# Overview

What is Mcu?

# 2. Mcu testing

2.1 Duty of testing side

2.2 Relative Knowledge before following up

BEART (clean generate parse build run report), EB tresos, PC PB.

2.3 Test Methodologies

How many test types in testing side

White-box, black-box, equivalent class, epd, multi-derivative, wiring, fault injection, performance, code coverage.

2.4 Run test step-by-step

Configuration, Initialize Mcu pins, look up interrupt vector table, change make file

2.5 Mcu test suites

Wiring, internal, external , equivalent class, multi-derivative, epd , fault injection, performance, code coverage test.

# 1. Overview

What is Mcu? What do you do to get basic knowledge?

☺ Please refer to:

<https://en.wikipedia.org/wiki/Microcontroller>

AUTOSAR\_SWS\_MCUDriver.pdf

# 2. Mcu Testing

**2.1 Duty of testing side**

Do you ever make a question that Developer also tests after coding, why we need Testers?

Tester stands in the different view than Developer. They do not trust the result of Developer that affected by self-view of them. Another view can see different things and find different bugs.

The different test types between Developer and Tester will be discussed in 2.3

**2.2 Relative Knowledge before following up**

**2.2.1 EB Tresos**

It’s the GUI that makes you configure data easier. After configuring, it generates \*.epc file.

**2.2.2 BEART**

Firstly, you must have the knowledge about BEART (clean generate parse build run report) which can be found in SOVVA book. It’s placed in \vnv\_dist\SOVVA\_doc\ sovva\_book.pdf in any view.

I can only take a brief detail about BEART:

+ Clean: remove all old files or make a new folder if it does not exist.

+ Generate: generate \*.c or \*.h files from \*.epc files, it picks data from \*.epc (configuration file) in each field and fill into \*.c or \*.h template files.

+ Parse: list all test suites and test cases in eunit.tst

+ Build: create obj files, link files, link variables,…

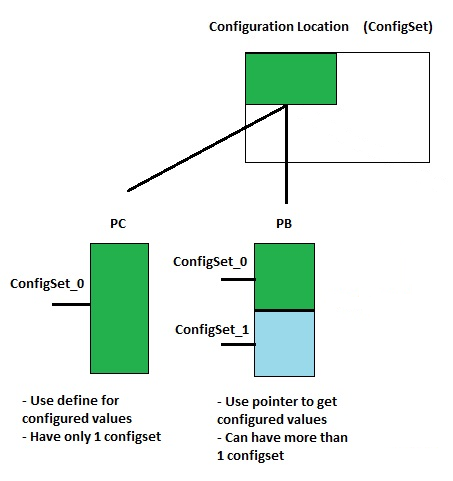
+ Run: load the code to hardware and run it, the result of EU\_ASSERT will be located in Test result location in RAM

+ Report: Get result from Test result location in RAM and create result.log file. Map between eunit.tst and result.log to determine which test case fail or not.

**2.2.3 PC, PB**

PC and PB relate to Mcu\_Init() function and the initial memory dedicated to configuration.

We can be had more than one configuration, we call it “ConfigSet”. The problem that how can we get the configured values from these ConfigSet. PC and PB give us the different ways to get these values.



**2.2.4 AUTOSAR**

Automotive Open System Architecture (AUTOSAR) defines what should you do and what must you do.

For example: Naming convention for APIs, and also for parameters; requirements that you must satisfy...

**2.3 Test Methodologies**

☺ Please refer to:

SMCAL\_Test\_Strategy.docx

We have many test types to cover all driver corner cases: BlackBox, WhiteBox, Equivalent Class (API, configuration, boundary), epd, multi-derivative, wiring, fault injection.

**2.3.1 BlackBox Test**

It uses only APIs that defined in AUTOSAR, set values for parameter and get the return results. We only use parameters, variables that defined in AUTOSAR, can know easily that developer can’t change these names. BlackBox test only test the reaction of APIs, whether APIs return true result or not.

**2.3.2 WhiteBox Test**

In contrast with BlackBox Test, WhiteBox affects deeply into APIs. It can change register values, create many dummy cases to examine APIs.

And, if you use variables that not defined in AUTOSAR (naming can be changed by developer), therefore it’s specific, and also call it’s WhiteBox.

**2.3.3 Equivalent Class API**

This test type aims to test many parameters of API. Especially, it uses many APIs together on one test case.

**2.3.4 Equivalent Class Configuration**

This test type aims to test with many values that configured in EB tresos. Many configured values

**2.3.5 Multi-Derivative Test**

Multi-Derivative test type has many sub-types:

* GENERATE\_SUCCESSFULLY
* GENERATE\_WITH\_ERROR
* COMPILE\_ONLY
* COMPILE\_WITH\_ERROR
* LINK\_WITH\_ERROR

***1/ GENERATE\_SUCCESSFULLY:***

It only aims to clarify that your configuration is true.

Clean: As normal.

Generate: Validate your configuration

Parse: Ignore this step

Build: Ignore this step

Run: Ignore this step

Report: As normal

***2/ GENERATE\_WITH\_ERROR***

Change \*.epc file with one or more wrong parameter (for example: out of parameter range). Your STDERR\_PATTERN must match with the Tresos message in cygwin.

Clean: As normal.

Generate: Validate your configuration, whether your STDERR\_PATTERN match with Tresos message or not

Parse: Ignore this step

Build: Ignore this step

Run: Ignore this step

Report: As normal

***3/ COMPILE\_ONLY***

Check that your test builds successfully.

Clean: As normal.

Generate: As normal.

Parse: As normal.

Build: Check that your test compiled without error.

Run: Ignore this step

Report: As normal

***4/ COMPILE\_WITH\_ERROR***

Change your test case with one or more error when compile (for example, call a function which is turned off in configuration). Please note that you must know about this error. Your STDERR\_PATTERN must match with Tresos message in Cygwin.

Clean: As normal.

Generate: As normal.

Parse: As normal.

Build: Validate your compile, whether your STDERR\_PATTERN match with Tresos message or not

Run: Ignore this step

Report: As normal

**For more information about Multi-Derivative, please refer to *Multiderivative tests - Freescale Romania* document.**

**2.3.6 Wiring Test**

This test type aims to verify whether all wiring connections are okay or not.

**2.3.7 Fault Injection**

If you do not have any idea about how to create other corner cases that can verify the driver, fault injection can help you.

**2.3.8 Performance test**

This test type aims to verify how many time does the function take? We use special define and special tool to run performance test. I will introduce later.

**2.3.9 Code coverage test**

This test type aims to examine how many driver code line do the test suites cover. To run 1 test suite for code coverage, it takes time much more than normal. Therefore, we can’t run all test suites for code coverage. We need create some code coverage test suites that use only for this test type, it may save time.

**2.4 Run test step-by-step**

**2.4.1 Configuration**

We configure in EB tresos. You must care of tresos version (14.2.1, 17.0.0, 19.0.0,…). About how to create 1 project, I don’t introduce.

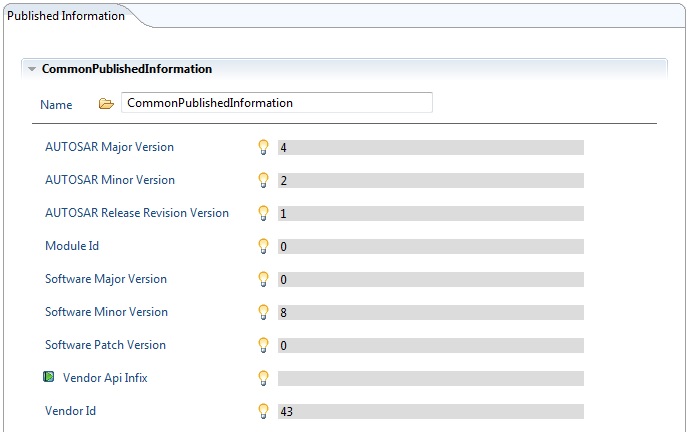
Before create new project, you must have all plugins that need to Mcu (Base, Dem, Det, Rte, Resource, Mcu may be Port in external test suite). For EB, you will add Base, Dem, Resource, Mcu to configure (may be Port in external test suite).

* ***Resource:***



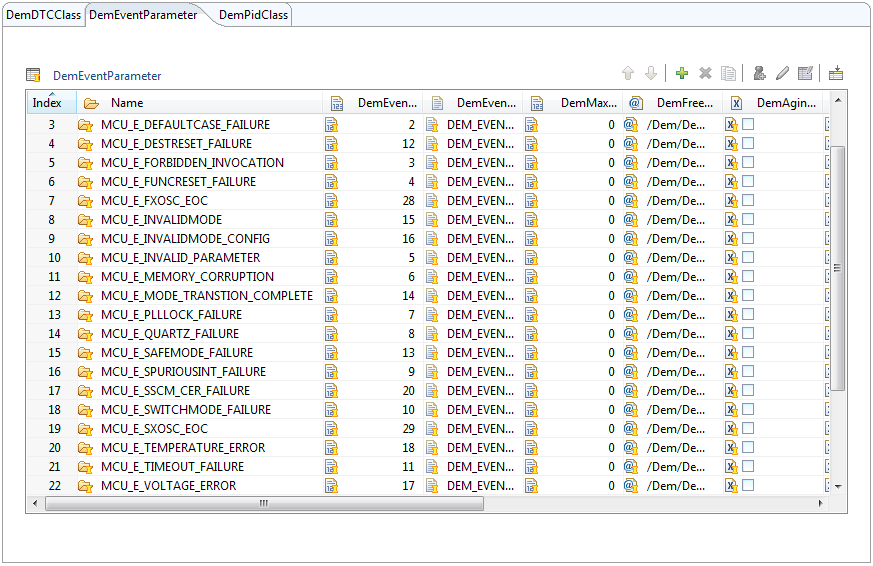
Only care about ResourceSubderivative, choose your sub-derivative of hardware.

* ***Base***



It just have information about AUTOSAR, release version,… Do not need any attempt to configure.

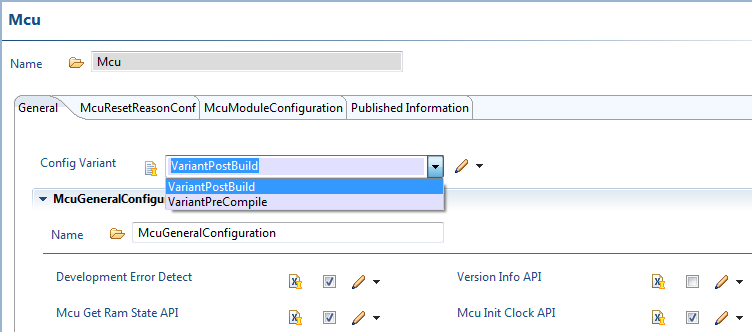
* ***Dem***



It has many things to configure (I recommend that you should ask everyone to configure this), but the most importance is DemEventParameter. At here, you add all Dem event that driver will send to upper layer.

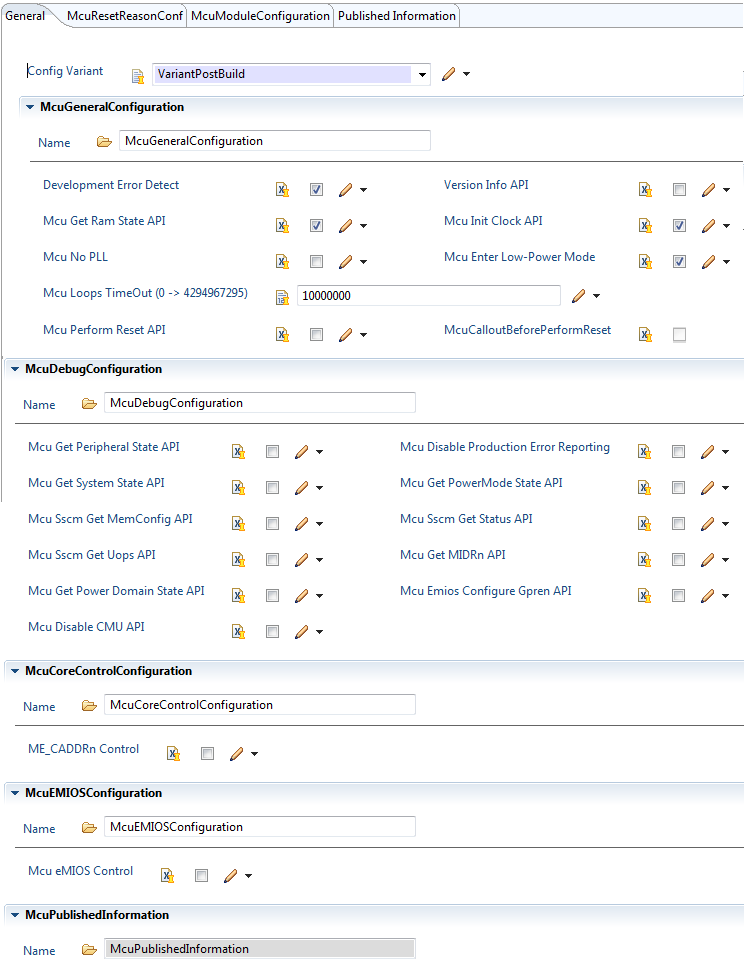
* ***Mcu***

+ Config variant:



Choose Config Variant PC, PB

+ Mcu General:



* Development Error Detection:

On/Off Det error report

* Version Info API:

On/Off Mcu\_GetVersionInfo() function, this Api provides the version information

of the MCU module

* Mcu Get Ram State API:

On/Off Mcu\_GetRamState() function, this Api provides the actual status of the microcontroller Ram

* Mcu Init Clock API:

On/Off Mcu\_InitClock() function, this Api shall initialize the PLL and other MCU

specific clock options. The clock configuration parameters are provided via the

configuration structure.

* Mcu No PLL:

ON: Don’t use PLL clock

OFF: Use PLL clock

* Mcu Enter Low-Power Mode:

ON: Use Low-Power Mode (Stop mode, Standby mode, HALT mode,…)

OFF: Don’t use Low-Power Mode (Only use DRUN mode, RUN mode,…)

* Mcu Perform Reset API:

On/Off Mcu\_PerformReset () function, this Api shall perform a microcontroller reset by

using the hardware feature of the microcontroller.

* Mcu Debug Configuration: Mcu Get Peripheral State API; Mcu Disable Production Error Reporting; Mcu Get System State API; Mcu Get PowerMode State API; Mcu Sscm Get MemConfig API; Mcu Sscm Get Status API; Mcu Sscm Get Uops API; Mcu Get MIDRn API; Mcu Get Power Domain State API; Mcu Emios Configure Gpren API; Mcu Disable CMU API.

These are options that permit On/Off the APIs of non-AUTOSAR.

* ME\_CADDRn Control:

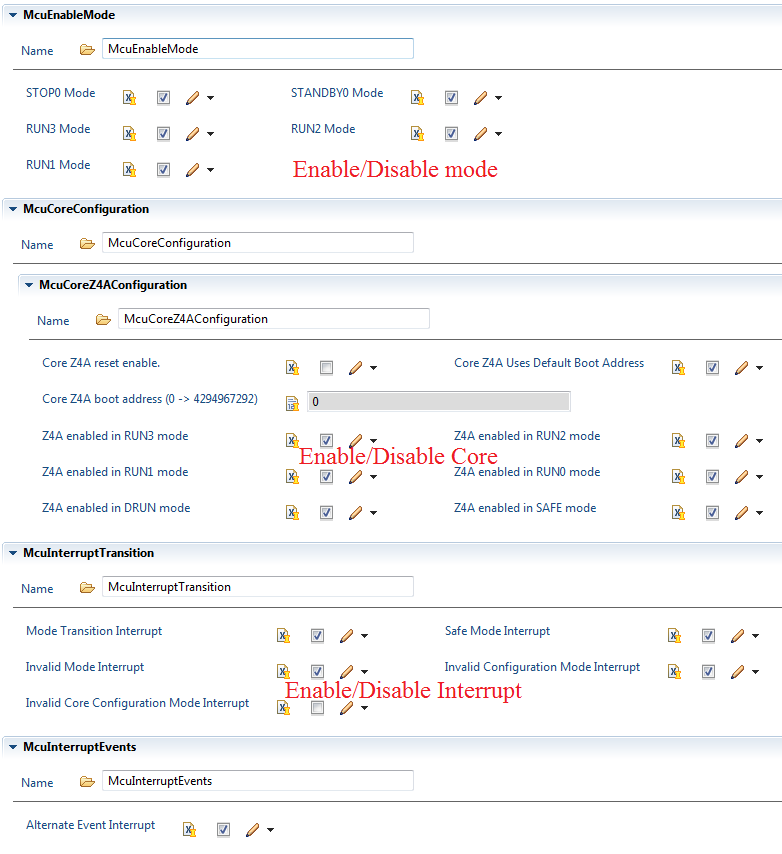
ON: Enable configurate for ME\_CADDRn registers (This register contain the boot address of each core and a bit for controlling whether the core is to be reset on the next mode change that has the core configured to be running in the target mode.

OFF: Disable configurate for ME\_CADDRn registers

* Mcu eMIOS Control:

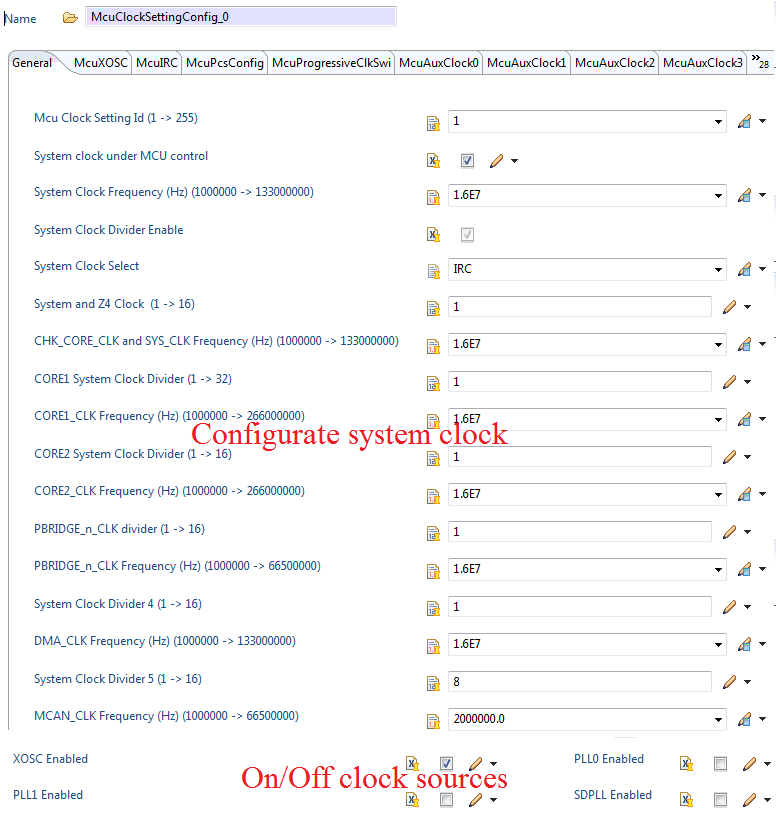
Enable/Disable configurate for McuEMIOS tap (in McuClockSettingConfig)

+ General in Mcu Module Configuration

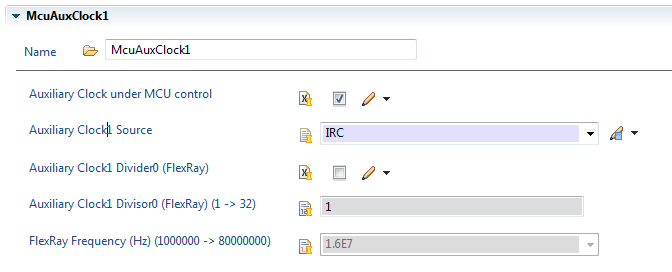


+ Cofigurate Clock

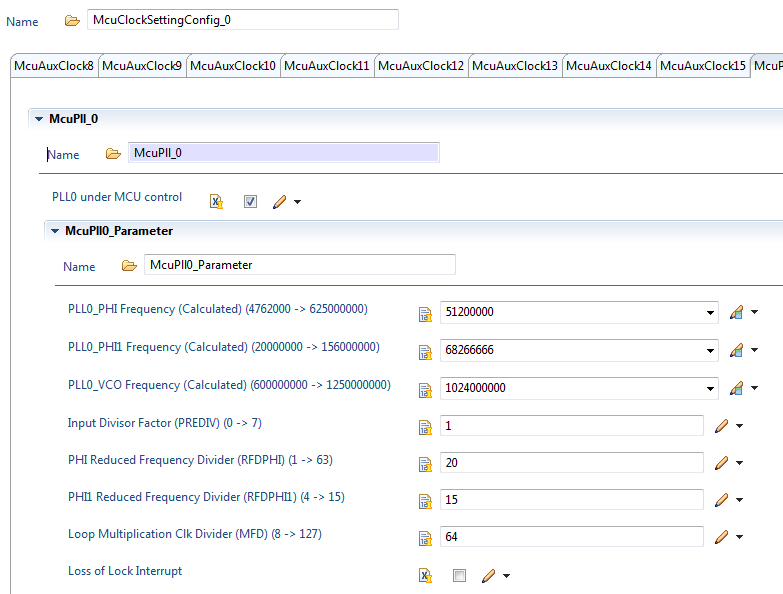
* General



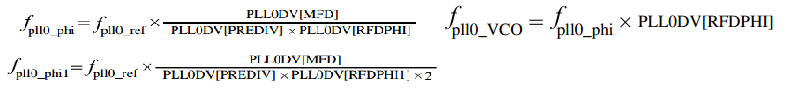
* Auxiliary Clock



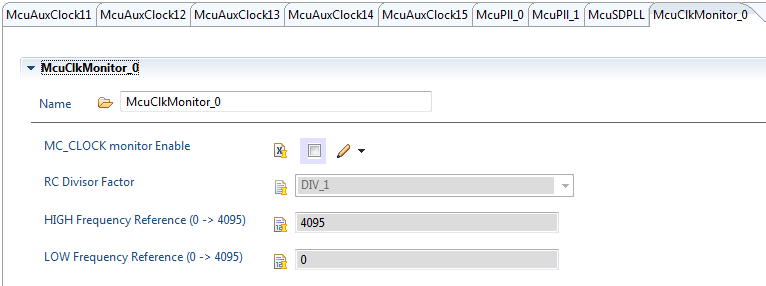
* PLL Clock



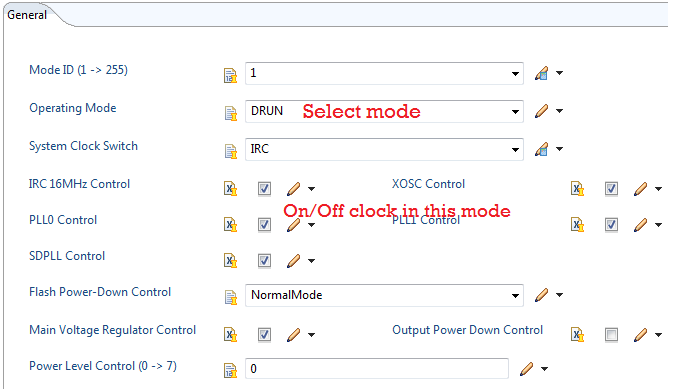
PLL clock is calculated according to the following equations: (Example on Racerunner platform):



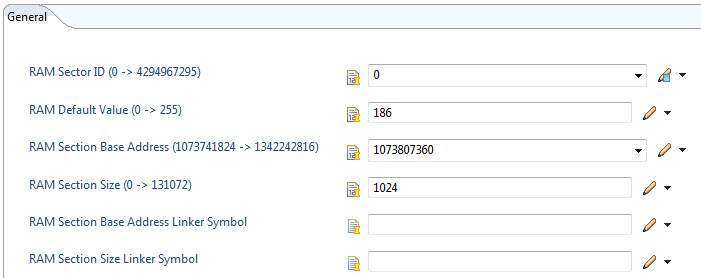
* Clock monitor



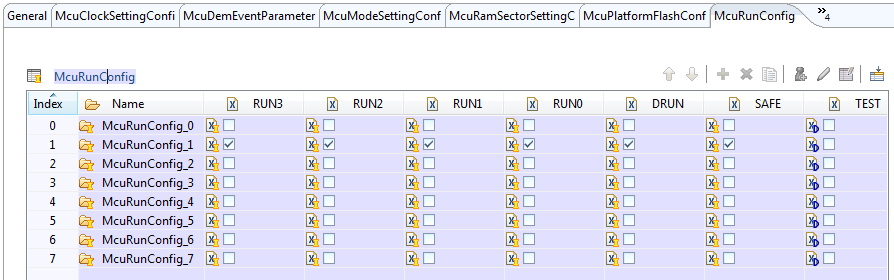
+ Cofigurate Mode



+ Mcu Ram Sector Setting Configirate

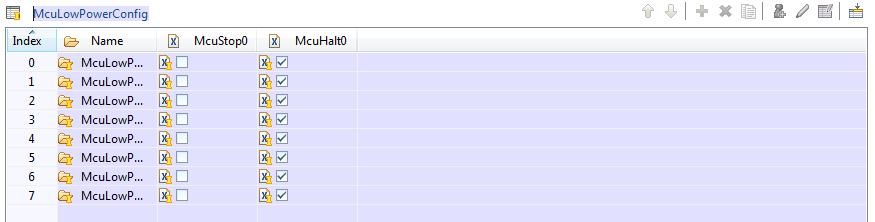


+ Mcu Run Config



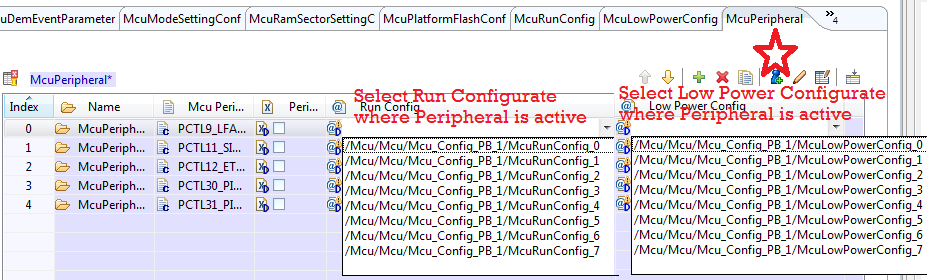
On/Off the modes (Run mode) in each McuRunConfig\_x

+ Mcu Low Power Config



On/Off the modes (Low Power Mode) in each McuLowPowerConfig\_x

+ Mcu Peripheral Configurate



With each platform, You can add all Peripherals by select red star. Which modes that the peripheral is active will be determined by its Run Config and Low Power Config.

**2.4.2 Initialize Mcu pins**

Mcu only use a pin to measure clock frequency. We have 2 steps to complete configure Mcu port pins:

* Select clock out pin

We must select clock out pin by use Evaluation Board document.

* Check the connection between mini module and mother board

Find: What does jump connect with clock out pin?

**2.4.3 Look up interrupt vector table**

We normally found the interrupt vector table in interrupt chapter in RM. List all Mcu vector and fill to specific\src\ Vector\_vle\_MCU\_00x.s. (for PPC) and fill to specific\src\ Mcu\_Test\_Setup.c (For ARM)

**2.4.4 Change make file**

Have you ever make a question that what is make file? What role is it?

Make file can know easily as path file. It list all paths of the needed files. It talk to compiler that “All file paths here, compiler, get it”.

Therefore, you need write down all file paths need to test in this file.

Some general information in make file: file paths (TC, TS, \*.epc, driver,…), config variant (PC, PB), core set (0,1,…)…

**2.5 Mcu test suites**

Depending on test methodologies I have referred in 2.3. Mcu test suites also classify base on test methodologies.

**2.5.1 Wiring test**

Currently, Mcu uses tse\_bbx\_wir\_mcu\_00100 as wiring test. Wiring test often measure external oscillator clock

**2.5.2 Internal Test**

Internal test is often the function tests. Internal test suite name is Mcu\_TS\_xxx and tsi\_bbx\_xxx\_mcu\_xxxxx. I just talk about some special test suites.

**2.5.3 External Test**

External test use MAF to measure all clock source (FXOSC; FIRC; PLL; …). External test suite name is tse\_bbx\_xxx\_mcu\_xxxxx

**2.5.4 BlackBox Test**

This test suite aims to test the interaction between other modules such as Resource, Port, Mcu,… Blackbox test suite name is tsx\_bbx\_xxx\_mcu\_xxxxx

**2.5.5 Equivalent Class**

Equivalent Class test suite name is Mcu\_TSx\_EqAPI\_xxx

**2.5.6 Multi-Derivative Test**

Multi-Derivative test suite name is Mcu\_TS\_M03 and Mcu\_TS\_M04. You can easily get knowledge about MD Test with pdf file I have recommended above.

**2.5.7 Epd Test**

Epd Test suite name is Mcu\_TS\_E01

**2.5.8 Fault Injection Test**

I will introduce somethings about Fault Injection Test. Its name is tsi\_bbx\_flt\_mcu\_xxxxx.

Mcu use Fault Injection Test: writen wrong value in Ram.

**2.5.9 Performance Test**

As above, this test types aim to measure function action time. Please take a look at Mcu\_TS\_PERF

**2.5.10 Code Coverage Test**

Test suite name is Mcu\_TS\_Cxx. It is internal test suite. Note that, we can’t add more test cases into test suite because code coverage test need more ram size than normal.