

Startup Code of the Test Program

Abstract

This application note describes how to link relevant program initialization code so-called startup code to the test program.

Table of contents

Abstract	1
1 Introduction.....	2
2 Test Program Structure	2
2.1 What is startup code?.....	2
2.2 Where to find the startup code?	3
2.3 What is a linker (script) file?	3
2.4 What linker (script) file is needed?	3
3 Implementation in TESSY	4
3.1 Target initialization code in function <code>main()</code>	4
3.2 Linking Multiple Board files.....	4
3.3 Linking an object file or a library.....	5
3.4 Linker (file) options	6
4 Troubleshooting.....	6
4.1 Compiler or linker errors.....	6
4.1.1 Invalid argument.....	7
4.1.2 Symbol not found	7
4.1.3 Memory areas overlap	7
4.2 Runtime errors.....	7
4.2.1 The test program gets stuck.....	8
4.2.2 Memory errors during runtime	9
4.2.3 Breakpoint cannot be set at <code>tslows_sync()</code>	9
4.2.4 Target Communication: Synchronization error.....	9

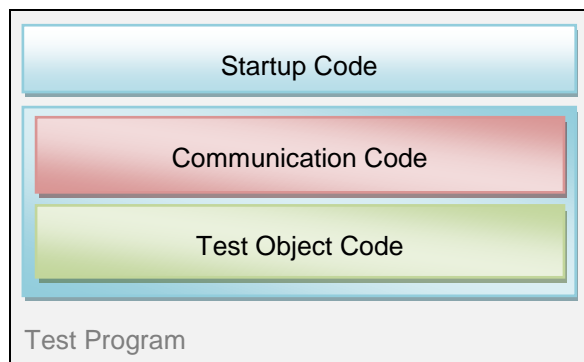
1 Introduction

The following chapters will give you an idea what startup code is about, where to find the startup code, and how to configure TESSY to integrate the startup code into your test program. *The document will not explain how to generate startup code.* Please consult your target development environment documentation for further information about this issue. Thus, *an executable project of your target development environment is required* in order to find the required files. Furthermore, a basic knowledge about TESSY's environment editor (TEE) is required before you work through this document. Please consult chapter 6.5 *TEE: Configuring the test environment* from TESSY's user manual if you have not done by now.

The document is divided into three major topics. The first part explains what startup code is about and where you might find it in your own target development project. The second part guides you through the different types of configurations that TESSY provides to integrate the startup code, and the third part helps you with some typical problems you might encounter while you try to integrate the startup code.

2 Test Program Structure

For the sake of clarity, we will divide the test program into three consecutive parts:



Unless you choose instrumentation for the test run, TESSY provides respectively generates the communication code and the corresponding basic configuration and Makefiles framework. The startup code as well as the test object code has to be provided by you respectively by the target development environment or by the manufacturer of the target board.

2.1 What is startup code?

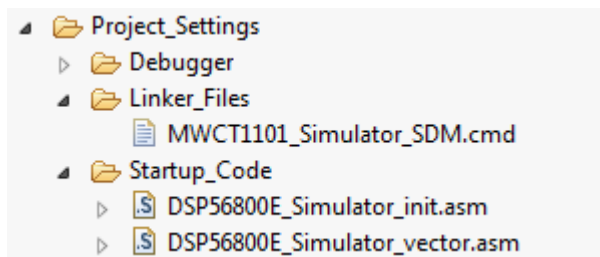
Startup code runs once at the very beginning of your program. It is used to setup the memory management, initialize registers, enable or disable interrupts and watchdogs and so on. Finally the startup code calls function `main()`. Some initializations such as interrupts and watchdog enabling or disabling might also be performed at the beginning of function `main()`. A so-called vector table file is used to setup interrupt functions.

Please note: Interrupts and watchdogs are not supported by TESSY and have to be disabled!

2.2 Where to find the startup code?

The easiest way to retrieve startup code is to setup a completely new project in your target environment for a simple example function. So that only one file contains your example function while all other files are most likely the startup code you will need.

Some development environments put the startup code into a special folder by default. The startup code is often split into different files which we call *board files*. So, in the project tree of your development environment you might search for a folder which contains files like `startup.c` or `cpu.c` or `io.c` or similar names. There may also be provided assembler files like `*_init.asm` as shown in the following example from CodeWarrior Development Studio's project view.



Other manufacturers provide a so-called board support library which contains those files. No matter what kind of board files is provided you should take a closer look into function `main()` of your development project.

```
38
39 int main(void)
40 {
41     const char test_string[]="hello\n";
42     int a, b, c;
43
44     DisableWatchdog;
45
```

Commands such as `DisableWatchdog`; have to be copied and transferred to TESSY as explained within the next chapter.

2.3 What is a linker (script) file?

The linker (script) file defines the sizes and addresses of the program's memory sections as well as the order of the object files and libraries to be linked.

2.4 What linker (script) file is needed?

Especially when using an instrumented test program TESSY's communication modules might require some more stack space. So you may copy the linker file from your target development project and set the stack size to about 200 bytes or more. Please consult the corresponding linker manual of the target development tools to

find out how this is done. If during the linking process some memory area is overlapping you may decrease TESSY's communication buffer size down to at least 8 bytes but it should not be lower than 16 bytes. It has to be a multiple of 8. The buffer size can be adjusted by the TEE attribute **Buffer Size**.

It is strongly recommended that you copy the linker file from your development project and adjust just it rather than creating a completely new linker file from scratch for TESSY. Try to link your development project's program with the adjusted linker file at first in order to check if it works fine. If the target program of your development environment does not work any longer with the adjusted linker file TESSY will also fail to run tests with the test program using the same linker file.

3 Implementation in TESSY

Due to the heterogeneous nature of the target development tools and also due to the ongoing improvement of TESSY's configuration there have evolved several different types of concepts on how to adapt the test program's compiling and linking requirements. This chapter will explain them step by step.

If the Makefile template has to be adjusted, please copy it into your TESSY project subfolder `tessy\config` and let the TEE attribute **Makefile Template** point to it. The reason is that a TESSY update or patch might exchange the installed Makefile template. This folder might also be a good place for your adjusted linker file.

3.1 Target initialization code in function `main()`

Some target initializations can be performed at the beginning of function `main()`. It depends on your target development tools if this is supported or even required. TESSY generally supports this feature by the two TEE attributes **Init Code** and **Init Definitions**. The contents of **Init Code** will be put literally at the beginning of function `main()`. The contents of **Init Definitions** will be put literally into the header section of the file which contains function `main()`. The latter might be useful to include some header files or to define external variables needed for the **Init Code**.

3.2 Linking Multiple Board files

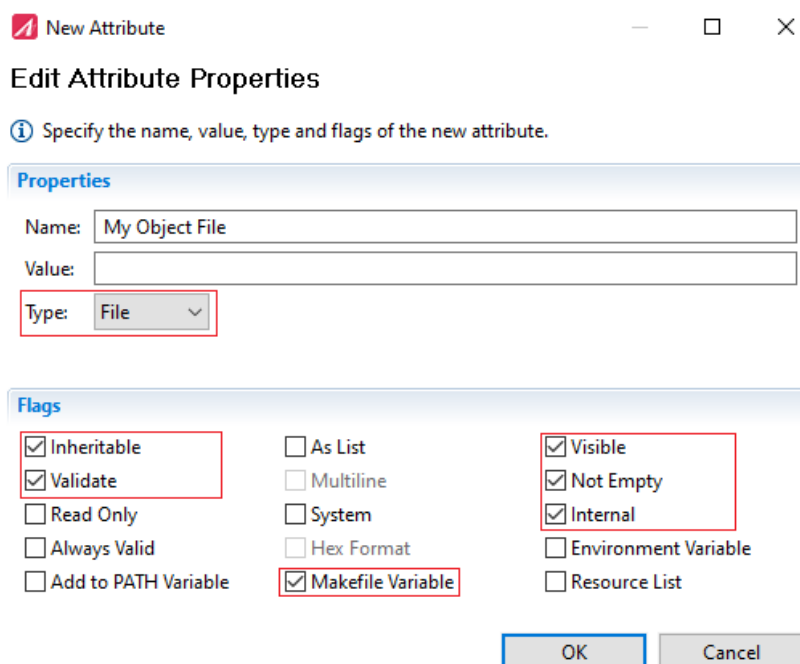
For those target development tools that require multiple board files to be linked TESSY provides for some compiler / target combinations a solution which can also easily be provided on request for other combinations.

Copy the board files into a separate folder you created for them in your TESSY project folder. The TEE attributes **InitSrcDir**, which points to the source folder of the board files, **InitObjDir**, which points to the object folder where the files from the source folder will be compiled to, and the attribute **Use Board Files**, which, if set to true, enables this feature, have to be present in the required TEE compiler / target combination and it has to be supported by the corresponding makefile template. The latest adaptations allow attribute **InitSrcDir** to be empty so that nothing is compiled but the files found in the folder pointed to by **InitObjDir** are linked to the test program if **Use Board Files** is set to true. The idea behind this feature is to allow the customer to just copy the precompiled board files from the target development project into the folder pointed to by **InitObjDir**. For several reasons **InitObjDir** should not point directly into the target development project. It is strongly recommended to create a

new folder for the board files in your TESSY project. So that you can archive the TESSY project folder and be sure to have all files together for future test runs.

3.3 Linking an object file or a library

Some target development tools require just one object file, e.g. a vector table file, or one board support library to be linked while the rest of the startup code is automatically linked by the linker itself. Some compiler / target combinations in the TEE contain an attribute called **Vector Table File** or **Library** which may be used to link the respective files. It is not sufficient to just create the desired attribute if it is missing in your TEE configuration because it has to be regarded by the Makefile template as well. If you want to add a file type attribute in the TEE the following attribute settings are recommended.



The variable will be generated into the Makefile if the **Makefile Variable** flag is set. The name of the variable in the generated Makefile is the uppercase version of the attribute's name while spaces are substituted by underscores. The **Internal** flag prevents you from accidentally removing the attribute. The attribute can then for example be referred to in the Makefile template as shown below.

```
$(MODULE_PATH_DOS)\ts_$(TESTOBJECT)_s$(EXEFILE_EXTENSION) : $(MODULE_PATH_DOS)\ts_$(TESTOBJECT)_s$(OBJFILE_EXTENSION) \
    $(S_SRC_OBJECTS) \
    $(START_OBJJS) \
    $(S_UC_OBJECT) \
    $(S_STUB_OBJECT) \
    $(S_OBJECTS)

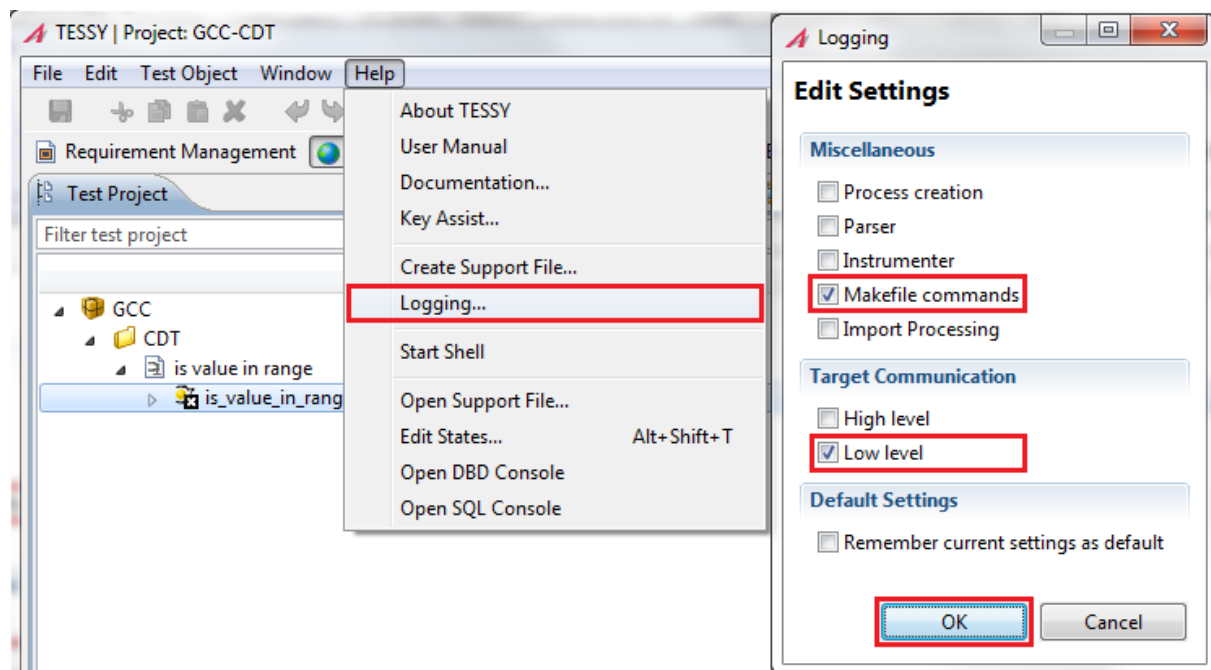
@echo *
@echo ***** Linking Slave *****
$(S_LINK) $(S_LINK_OPTIONS) $(USER_LINK_OPTIONS) \
    $(MY_OBJECT_FILE) $(S_OBJECTS) $(MODULE_PATH_DOS)\ts_$(TESTOBJECT)_s$(OBJFILE_EXTENSION) \
    $(S_SRC_OBJECTS) $(START_OBJJS) $(S_UC_OBJECT) $(S_STUB_OBJECT) \
    -o "$@"
@-ts_rm "$(MODULE_PATH)"/*$(OBJFILE_EXTENSION)
```

3.4 Linker (file) options

Some Makefile templates generate a new linker file based on a given template linker file. Others add linker arguments to the linker command line. Generally, the TEE attribute **Linker Options** may be used to add necessary linker arguments but for some compiler / target combinations there are already special TEE attributes available which should be preferred to set the corresponding values, for instance **Code Segment Begin**, **Entry Point**, or **Stack Size**. So please watch carefully all non-expert-mode TEE attributes which are provided for your compiler / target combination for these are usually very important for a successful test run. Please consult the specific TESSY application note for your compiler / target combination for further information.

4 Troubleshooting

The troubleshooting chapter is divided into compiler or linker errors and runtime errors. It is recommended to turn on TESSY's logging at first as shown below if you encounter a compile respectively link time or runtime error. Then try to reproduce the error by compiling and linking or running the test program again.

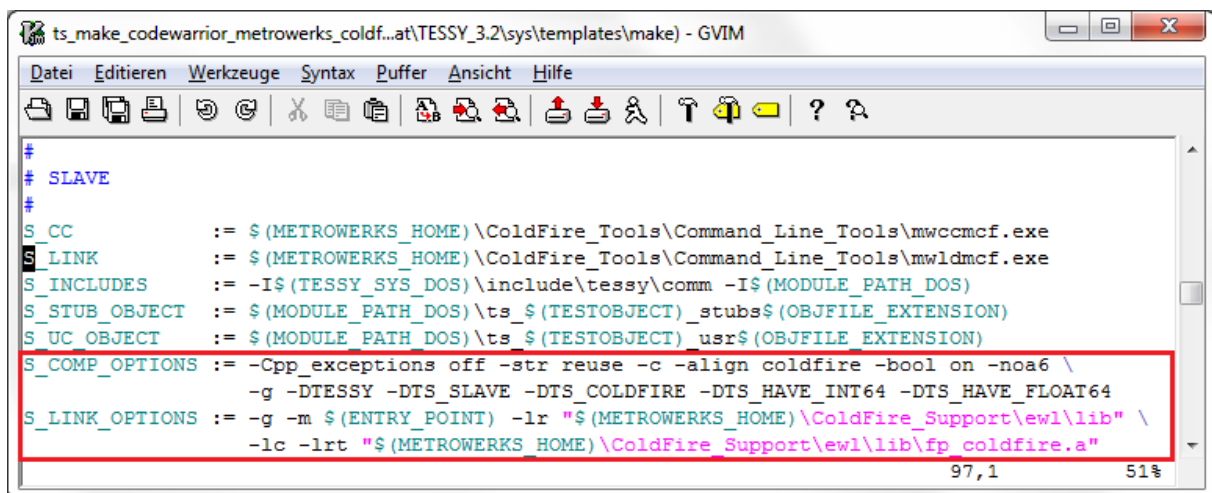


4.1 Compiler or linker errors

Generally, most problems concerning compiler or linker errors originate from wrong or missing startup code or from inappropriate linker files. TESSY's compiler / target combinations have been tested with at least one concrete compiler, debugger, and processor. However, because to the huge amount of processors is it very likely that you have to adjust your TESSY configuration, i.e. choose special startup code and linker files if your target development tools require it (see 3).

4.1.1 Invalid argument

If the compiler call issues an *invalid argument* you should check the compiler arguments your target development environment uses and compare them with the ones you find in your TESSY Makefile template. Feel free to adjust the Makefile variable `S_COMP_OPTIONS` to change the compiler arguments but keep all defines beginning with "TS_". Variable `S_LINK_OPTIONS` may be adjusted to change the linker arguments. It is strongly recommended to keep the debugging arguments for both compiler and linker arguments. Otherwise parts of TESSY's communication code might be missing in the test program.



```
#
# SLAVE
#
S_CC      := $(METROWERKS_HOME)\ColdFire_Tools\Command_Line_Tools\mwccmcf.exe
S_LINK    := $(METROWERKS_HOME)\ColdFire_Tools\Command_Line_Tools\mwldmcf.exe
S_INCLUDES := -I$(TESSY_SYS_DOS)\include\tessy\comm -I$(MODULE_PATH_DOS)
S_STUB_OBJECT := $(MODULE_PATH_DOS)\ts_$(TESTOBJECT)_stubs$(OBJFILE_EXTENSION)
S_UC_OBJECT := $(MODULE_PATH_DOS)\ts_$(TESTOBJECT)_usr$(OBJFILE_EXTENSION)
S_COMP_OPTIONS := -Cpp_exceptions off -str reuse -c -align coldfire -bool on -noa6 \
-g -DTESSY -DTS_SLAVE -DTS_COLDFIRE -DTS_HAVE_INT64 -DTS_HAVE_FLOAT64
S_LINK_OPTIONS := -g -m $(ENTRY_POINT) -lr "$(METROWERKS_HOME)\ColdFire_Support\ewl\lib" \
-lc -lrt "$(METROWERKS_HOME)\ColdFire_Support\ewl\lib\fp_coldfire.a"
```

4.1.2 Symbol not found

It depends on the specific file which requires a symbol. For instance, `ts_src01.o` contains your test object. That means that a header file or library might be missing or is not defined. So please check TESSY's Test Interface Editor if all variables have been defined. Check your include paths and so on.

However, if the symbol is missing in one of your board files you should add more board files from your target development environment. It is also recommended to put the header files into the folder point to by `InitSrcDir`, if you use this attribute, as well (see 3.2). Too few, missing, or even wrong startup code results usually in runtime errors (see 4.2).

4.1.3 Memory areas overlap

Some target devices have very little memory. So, at first you might try to reduce TESSY's communication buffer which can be adjusted by the TEE attribute **Buffer Size** (see 2.4). If this is not sufficient you will have to adjust your linker file. Please consult the linker's manual or ask the developer of the test object you want to test or ask the manufacturer of the device.

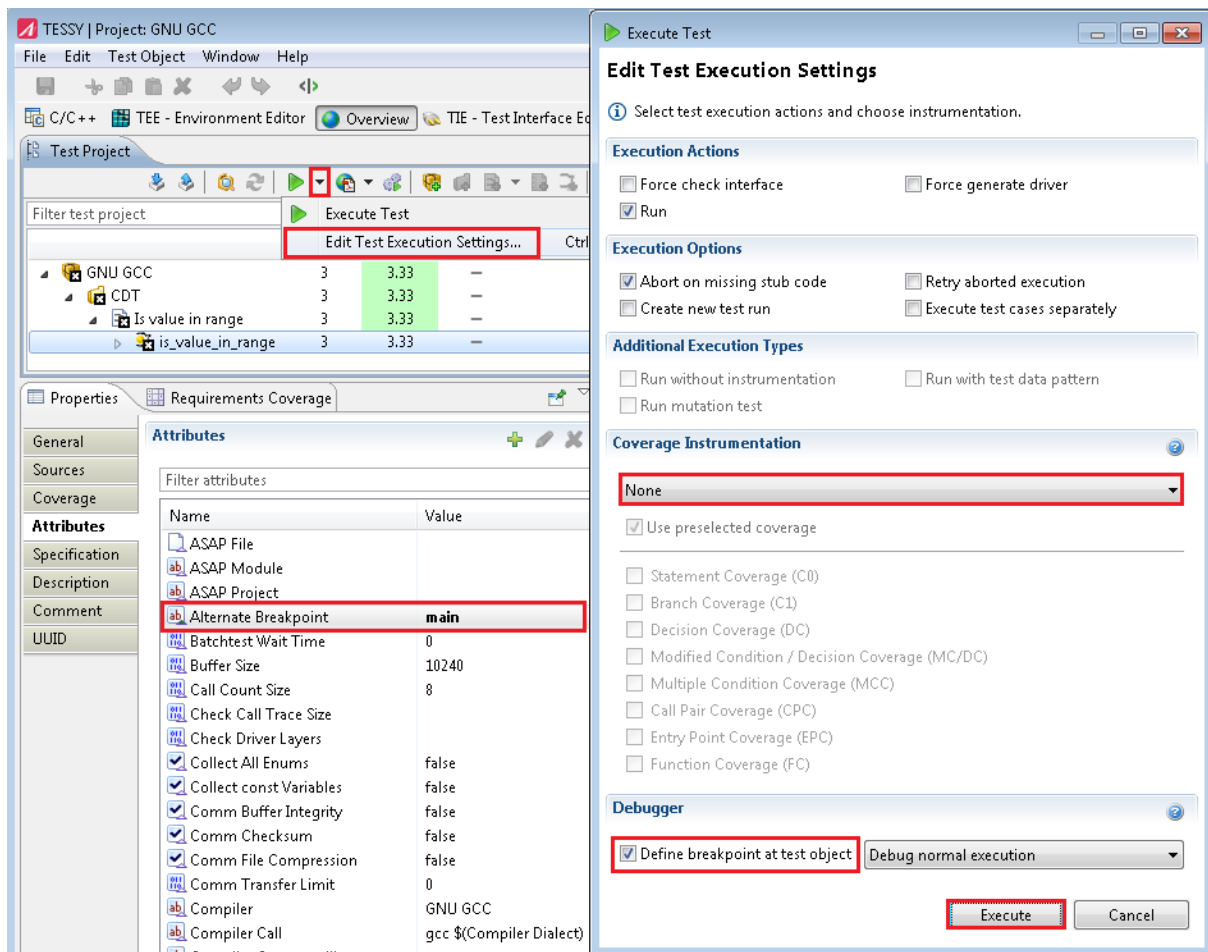
4.2 Runtime errors

You need a bit of investigative instinct to track the problem. A good choice to start with is to set an alternate breakpoint and start the test run as shown below. The

target debugger should halt at function `main()` . If it does not halt at `main()` than the startup code is wrong or missing or the linker file is wrong (see 2.1 and 2.3).

If `main()` can be reached you should step through the code until function `tslows_sync()` is called. If the function cannot be reached than again the startup code is wrong or missing or the linker file is wrong (see 2.1 and 2.3). If the function `tslows_sync()` is reachable by stepping from `main()` on to it than the startup code and linker file are most likely correct and you may run a check on the driver layers as described in application note *048 Using Test Driver Communication Tests*.

If, however, the debugger could not even load the binary you should try to load it manually into the target debugger to find the problem.



4.2.1 The test program gets stuck

At first you should try to debug your test object interactively by choosing **Define Breakpoint at Test Object** from the **Execute Test** dialog. If the debugger does not halt at your test object follow the hints in 4.2.

4.2.2 Memory errors during runtime

Follow the hints in 4.2. Depending on the debugger adaption you should check if the communication buffer(s) are writable, i.e. there are either two of them (`ts_input_buffer`, `ts_output_buffer`) or one (`ts_exchange_buffer`), and a flag variable (`ts_sync_state`) that should all be writable as well as `TS_CURRENT_TESTCASE` and `TS_CURRENT_TESTSTEP`. If they are not writable you have to check your linker file.

4.2.3 Breakpoint cannot be set at `tslows_sync()`

Most likely the compiler has removed the function at compile time because it seems to do nothing relevant. So, you have to check your Makefile template and see if you used optimization arguments or missed debug arguments.

4.2.4 Target Communication: Synchronization error

Please check the endianness which can be altered by the TEE attribute **Endianness**. If the endianness is correct follow the hints in 4.2.