MPLAB® Harmony Help - Test Harness Library

MPLAB Harmony Integrated Software Framework v1.11

Test Harness Library

This section describes the Test Harness Library that is available in MPLAB Harmony.

Description

There are two types of test libraries included in the MPLAB Harmony installation. The first type is a single Test Harness Library. This library implements a state machine that runs and manages specially developed tests that are used to validate the behavior of MPLAB Harmony-compatible libraries in the different execution environments supported. The second type of test library implements the tests that are run by the Test Harness Library. These libraries are predefined and implemented by Microchip to validate existing MPLAB Harmony drivers, middleware, and other libraries. This type of library may be used in conjunction with the test harness library to verify the behavior of different implementations of existing MPLAB Harmony libraries.

Introduction

This library runs and manages specially developed tests used to validate the behavior of MPLAB Harmony-compatible libraries in the different execution environments supported.

Description

This Test Harness Library help and usage documentation defines the library's interface and expected behavior. To utilize the Test Harness Library, a test application must be developed that initializes and runs the Test Harness Library. Additionally, tests must be written that fulfill the requirements of the Test Harness Library. These tests need to call the interface functions of a Library Under Test (LUT) and verify its expected behavior. Refer to the MPLAB Test Harness User's Guide for additional details on developing tests and test applications.

The Test Harness Library can execute a number of tests, each of which validates some aspect of the interface and behavior of a MPLAB Harmony-compatible library. Each test can have a number of sub-tests. If all sub-tests pass, the test passes. If any sub-test fails, the test fails. If all tests in a given configuration pass, the harness reports an overall passing result. If any test fails, the harness reports an overall failure result. Error messages, individual test results, and other messages are output as they occur, provided that a SYS_DEBUG output method is available and has been configured. Otherwise, a debugger must be used to check the results stored in an internal data structure of the Test Harness.

Library Overview

This section provides an overview of the Test Harness Library.

Description

The library interface routines are divided into various sub-sections, which address one of the blocks of the overall operation of the Test Harness Library.

Library Interface Section	Description
System Interface Functions	Functions used by the MPLAB Harmony system to initialize and maintain the Test Harness Library and libraries under test.
Platform Initialization Functions	Functions used to perform hardware or test platform-specific initialization.
Test Results Functions	Functions used by tests to report results to the harness.
Test Time-out Functions	Function used to maintain the test time out timer.
Test Harness Initialization Data Types	These structures provide the list of tests that the harness executes.
Test Interface Function Pointer Types	These pointers define the functions that the Test Harness requires each test must implement.

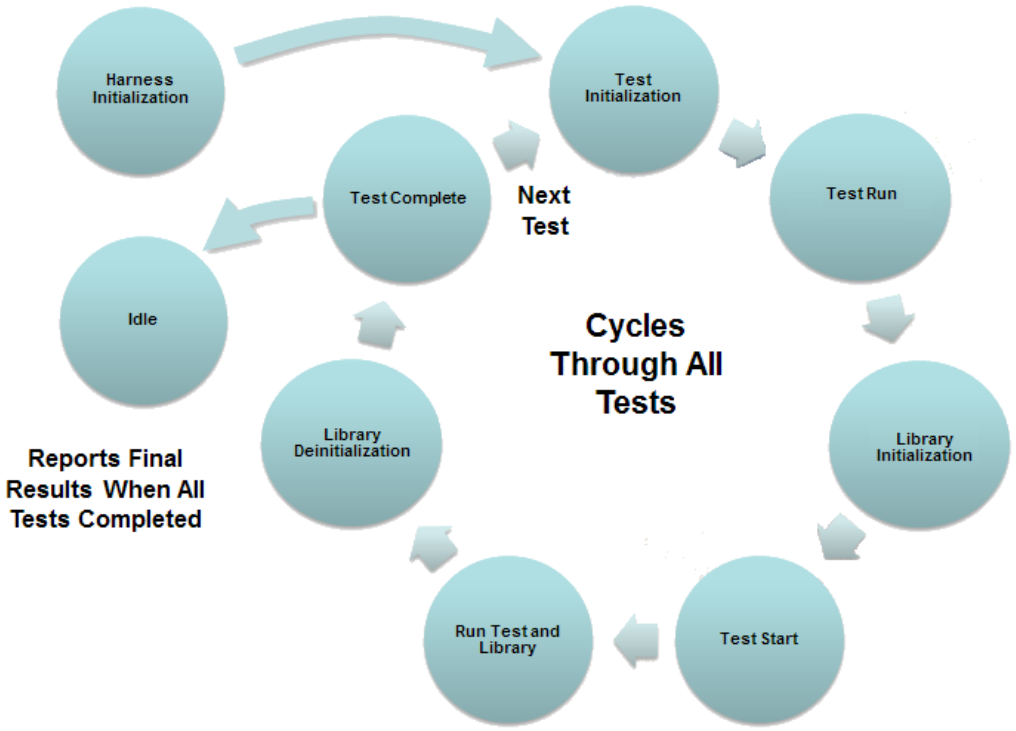
State Machine

This topic describes the State Machine for the MPLAB Harmony Test Harness Library.

Description

The Test Harness State Machine (see **Figure 1**) runs in a cycle that executes all tests in the test list that was provided to it when it was initialized. This state machine first performs any required configuration-specific initialization by calling the TEST_PlatformInitialize function. Then, it initializes and prepares the first test to be run. When the test indicates that it is ready, the harness initializes the library and runs both the test and the LUT. As the test runs, it reports intermediate "sub-test" results back to the harness, which accumulates them for later reporting. The tests and harness use debug message output to print results and information, if so configured. Intermediate debug messages can be "printed" at any time, but they may affect test timing. When the test is completed, the test harness deinitializes the LUT, reports results for that test, and advances to the next test library to start the process over again. When the last test finishes, the harness reports the final results and goes idle or hits a breakpoint (in Debug mode).

Figure 1: Test Harness State Machine



Refer to the MPLAB Harmony Test Harness User's Guide for information on using the Test Harness Library to develop test applications and for information on developing tests that meet the requirements of the test harness.

Using the Library

This topic describes the basic architecture of the Test Harness Library and provides information and examples on its use.

Description

The interface to the Test Harness Library is defined in the `test_harness.h` header file.

Any C language source (`.c`) file that uses the Test Harness Library must include `test_harness.h`.

How the Library Works

This section provides information on how the Test Harness Library works.

Core Functionality

Refer to the MPLAB Harmony Test Harness User's Guide for information on the core functionality of the Test Harness Library.

Configuring the Library

Macros

	Name	Description
	TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY	Defines the maximum number of "Tasks" routines implemented by a single library under test.
	TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY	Defines the maximum number of instances of a single library under test.
	TEST_IDLE_SLEEP_MS	Defines the number of milliseconds the test harness will sleep when the test reports that it is idle.
	TEST_IDLE_SLEEP_MS_LIBRARY	Defines the number of milliseconds the library tasks function will sleep when the library is idle.
	TEST_TIMER_CLOCK_PRESCALER	Defines the selected clock prescaler value for the timeout timer.
	TEST_TIMER_CLOCK_SOURCE	Defines the selected clock source for the timer used for the timeout timer.
	TEST_TIMER_ID	Defines the timer peripheral used for the test timeout timer.
	TEST_TIMER_INCREMENT_PERIOD	Defines the number of timer counts that make up a single timer period.
	TEST_TIMER_INTERRUPT_PRIORITY	Defines the interrupt priority for the selected test timer.
	TEST_TIMER_INTERRUPT_SOURCE	Defines the interrupt source ID of the timer used for the timeout timer.
	TEST_TIMER_INTERRUPT_SUBPRIORITY	Defines the interrupt subpriority for the selected test timer.
	TEST_TIMER_INTERRUPT_VECTOR	Defines the interrupt vector for the selected test timer.
	TEST_TIMER_MS_PER_INCREMENT	Defines the number of milliseconds per timer increment period.
	TEST_TIMER_MS_TIMEOUT	Defines the number of milliseconds that must expire before a test will time out.

Description

The configuration of the Test Harness Library is based on the file `system_config.h`.

This header file contains the configuration selections for the Test Harness Library. Based on the selections made, the library may support the support selected features.

The path to this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY Macro

Defines the maximum number of "Tasks" routines implemented by a single library under test.

File

test_config_template.h

C

```
#define TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY 5
```

Description

Maximum Number of Tasks for a Library Under Test

This configuration parameter defines the maximum number of "Tasks" routines are allowed for a single library under test.

Remarks

This parameter has a default value defined in test_harness.h that will generate a warning when used. To eliminate the warning, define a value in system_config.h.

TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY Macro

Defines the maximum number of instances of a single library under test.

File

test_harness.h

C

```
#define TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY 3
```

Description

Maximum Number of Instances of a Library Under Test

Remarks

Refer to config/test_config_template.h for additional information.

TEST_IDLE_SLEEP_MS Macro

Defines the number of milliseconds the test harness will sleep when the test reports that it is idle.

File

test_config_template.h

C

```
#define TEST_IDLE_SLEEP_MS 50
```

Description

Test Idle Sleep Time

This configuration option defines the number of milliseconds that the test harness will sleep before calling the tasks function of the test when the it reports it is idle.

Remarks

This capability is only supported in an RTOS configuration, but this value must be defined in all configurations.

TEST_IDLE_SLEEP_MS_LIBRARY Macro

Defines the number of milliseconds the library tasks function will sleep when the library is idle.

File

test_config_template.h

C

```
#define TEST_IDLE_SLEEP_MS_LIBRARY 5
```

Description

Library under Test Idle Sleep Time

This configuration option defines the number of milliseconds that the test harness will sleep before calling the tasks function of the library under test when the library status reports it is idle.

Remarks

This capability is only supported in an RTOS configuration, but this value must be defined in all configurations.

TEST_TIMER_CLOCK_PRESCALER Macro

Defines the selected clock prescaler value for the timeout timer.

File

test_config_template.h

C

```
#define TEST_TIMER_CLOCK_PRESCALER TMR_PRESCALE_VALUE_256
```

Description

Timeout Timer Clock Prescaler

This configuration option defines the selected clock prescaler value for the timeout timer.

Remarks

The frequency of the clock source selected by the TEST_TIMER_CLOCK_SOURCE is divided by the value identified by this option.

The test harness takes direct ownership of this timer, utilizing the timer peripheral library to control it. The selected prescaler value must be valid for the selected timer, as defined by the timer peripheral library.

TEST_TIMER_CLOCK_SOURCE Macro

Defines the selected clock source for the timer used for the timeout timer.

File

test_config_template.h

C

```
#define TEST_TIMER_CLOCK_SOURCE TMR_CLOCK_SOURCE_PERIPHERAL_CLOCK
```

Description

Timeout Timer Clock Source

This configuration option defines the selected clock source for the timer used for the timeout timer.

Remarks

The test harness takes direct ownership of this timer, utilizing the timer peripheral library to control it. The selected clock source must be valid for the selected timer, as defined by the timer peripheral library.

TEST_TIMER_ID Macro

Defines the timer peripheral used for the test timeout timer.

File

test_config_template.h

C

```
#define TEST_TIMER_ID TMR_ID_1
```

Description

Timeout Timer

This configuration option defines the timer peripheral used for the test timeout timer. It must be defined as the timer peripheral library instance ID of the desired timer instance.

Remarks

This parameter has a default value defined in test_harness.c that will generate a warning when used. To eliminate the warning, define a value in system_config.h.

The test harness takes direct ownership of this timer, utilizing the timer peripheral library to control it. The selected timer ID must be valid, as defined by the timer peripheral library.

TEST_TIMER_INCREMENT_PERIOD Macro

Defines the number of timer counts that make up a single timer period.

File

test_config_template.h

C

```
#define TEST_TIMER_INCREMENT_PERIOD 31250
```

Description

Test Timer Increment Period

This configuration option defines the number of timer counts of the selected timer clock source that will be counted before a single timer period expires.

Remarks

The test harness takes direct ownership of this timer, operating it in 16-bit mode. The selected period value must be valid for the selected timer, as defined by the timer peripheral library.

TEST_TIMER_INTERRUPT_PRIORITY Macro

Defines the interrupt priority for the selected test timer.

File

test_config_template.h

C

```
#define TEST_TIMER_INTERRUPT_PRIORITY INT_PRIORITY_LEVEL1
```

Description

Test Timer Interrupt Priority

This configuration option defines the interrupt priority for the selected test timer.

Remarks

None.

TEST_TIMER_INTERRUPT_SOURCE Macro

Defines the interrupt source ID of the timer used for the timeout timer.

File

test_config_template.h

C

```
#define TEST_TIMER_INTERRUPT_SOURCE INT_SOURCE_TIMER_1
```

Description

Timeout Timer Interrupt Source

This configuration option defines the interrupt source ID of the timer used for the timeout timer.

Remarks

The test harness utilizes the interrupt system service to control this interrupt. The selected interrupt source must be valid for the selected timer, as defined by the interrupt system service.

TEST_TIMER_INTERRUPT_SUBPRIORITY Macro

Defines the interrupt subpriority for the selected test timer.

File

test_config_template.h

C

```
#define TEST_TIMER_INTERRUPT_SUBPRIORITY INT_SUBPRIORITY_LEVEL0
```

Description

Test Timer Interrupt Subpriority

This configuration option defines the interrupt subpriority for the selected test timer.

Remarks

None.

TEST_TIMER_INTERRUPT_VECTOR Macro

Defines the interrupt vector for the selected test timer.

File

test_config_template.h

C

```
#define TEST_TIMER_INTERRUPT_VECTOR INT_VECTOR_T1
```

Description

Test Timer Interrupt Vector

This configuration option defines the interrupt vector for the selected test timer.

Remarks

None.

TEST_TIMER_MS_PER_INCREMENT Macro

Defines the number of milliseconds per timer increment period.

File

test_config_template.h

C

```
#define TEST_TIMER_MS_PER_INCREMENT 100
```

Description

Test Timer Milliseconds Per Increment

This configuration option defines the number of milliseconds that expire per timer period. The internal timeout timer is incremented by this many milliseconds every time the timer period defined by the TEST_TIMER_INCREMENT_PERIOD configuration option expires.

Remarks

This configuration option effectively translates the timer period from scaled clock cycles to milliseconds.

TEST_TIMER_MS_TIMEOUT Macro

Defines the number of milliseconds that must expire before a test will time out.

File

test_config_template.h

C

```
#define TEST_TIMER_MS_TIMEOUT 2000
```

Description

Test Timer Timeout Value

This configuration option defines the number of milliseconds that must expire before a test will time out and be considered as a failure.

Remarks

None.

Building the Library

This section lists the files that are available in the Test Harness Library.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/framework/test`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/test_harness.h`	This file provides the interface definitions of the Test Harness Library.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`/src/test_harness.c`	Test Harness Library implementation.

Optional File(s)

There are no optional files for the Test Harness Library.

Module Dependencies

The Test Harness Library is dependent upon the following libraries and header files.

- Interrupt System Service Library
- Operating System Abstraction Layer (OSAL)
- Debug System Service Library (or an implementation of the SYS_DEBUG_PRINT and SYS_DEBUG_MESSAGE macros)
- sys_common.h
- sys_module.h

Library Interface

a) System Interface Functions

	Name	Description
	TEST_Initialize	MPLAB Harmony test harness initialization routine.
	TEST_Tasks	Runs the current test's tasks functions.
	TEST_HarnessTasks	MPLAB Harmony test harness' tasks function.
	TEST_LibraryTasksPolled	Calls the a tasks function of a library under test when the test harness is ready to run the library.
	TEST_LibraryTasksISR	Calls an ISR-based tasks function of the library under test.

b) Platform Initialization Functions

	Name	Description
	TEST_PlatformInitialize	Called by the test harness to initialize configuration-specific test support.
	TEST_PlatformReady	Indicates that the test configuration is ready and testing can begin.

c) Test Results Functions

	Name	Description
	TEST_HasCompleted	Allows a test to indicate that it has completed.
	TEST_HasPassedSubtest	Allows a test module to indicate that an individual subtest has completed and identifies if it has passed or failed.
	TEST_LibraryInitialize	Indicates that the test is ready for the harness to initialize the library under test.

d) Test Time-out Functions

	Name	Description
	TEST_TimerIncrement	Increments test timeout timer.

e) Test Harness Initialization Data Types

	Name	Description
	LIBRARY_UNDER_TEST	This is type LIBRARY_UNDER_TEST.
	_library_under_test	This is type LIBRARY_UNDER_TEST.
	_test_data	Holds data necessary to initialize and run a single test.
	_test_init_data	Holds the test harness's initialization data.
	TEST_DATA	Holds data necessary to initialize and run a single test.
	TEST_INIT_DATA	Holds the test harness's initialization data.

f) Test Interface Function Pointer Types

	Name	Description
	TEST_MODULE_INITIALIZE_ROUTINE	Pointer to a routine that initializes a test module.
	TEST_MODULE_START_FUNCTION	Pointer to a routine that starts the test module.
	TEST_MODULE_STATUS_ROUTINE	Pointer to a routine that gets the current status of a test module.
	TEST_MODULE_TASKS_ROUTINE	Pointer to a routine that performs the tasks necessary to maintain the state machine of a test module system module.

Description

This section describes the APIs of the Test Harness Library.

a) System Interface Functions

TEST_Initialize Function

MPLAB Harmony test harness initialization routine.

File

test_harness.h

C

```
void TEST_Initialize(const TEST_INIT_DATA * init);
```

Returns

None.

Description

This function initializes the MPLAB Harmony test harness. It places the harness in its initial state and prepares it to run so that the TEST_HarnessTasks function can be called.

Remarks

This routine must be called from the SYS_Initialize function.

The test harness will initialize tests and libraries under test later, under control of its state machine.

Preconditions

All other system required initialization routines should be called before calling this routine (in "SYS_Initialize"). This function should be called before or in place of the application's in initialization function.

Example

```
TEST_Initialize();
```

Parameters

Parameters	Description
init	Pointer to test initialization data.

Function

void TEST_Initialize (const APP_TH_INIT_DATA *init)

TEST_Tasks Function

Runs the current test's tasks functions.

File

test_harness.h

C

```
void TEST_Tasks(unsigned int number);
```

Returns

None.

Description

This function runs the current test's tasks functions, once the harness determines that the test should run.

Remarks

This function uses the following configuration parameters.

- TEST_IDLE_SLEEP_MS - Number of milliseconds that the test will sleep waiting for the module when it is busy.

Preconditions

The system and test harness must have all been initialized.

Example

None.

Parameters

Parameters	Description
number	Test's tasks function number. (Tests can have more than one tasks function.

Function

void TEST_Tasks(unsigned int number)

TEST_HarnessTasks Function

MPLAB Harmony test harness' tasks function.

File

test_harness.h

C

```
void TEST_HarnessTasks();
```

Returns

None.

Description

This routine is the tasks function of the MPLAB Harmony test harness and defines the state machine and core logic of the test harness.

Remarks

This routine must be called from SYS_Tasks() routine or from an RTOS thread loop.

Preconditions

Any required system initialization functions must have been called and the "TEST_Initialize" function must have been called.

Example

```
TEST_HarnessTasks();
```

Function

void TEST_HarnessTasks (void)

TEST_LibraryTasksPolled Function

Calls the a tasks function of a library under test when the test harness is ready to run the library.

File

test_harness.h

C

```
void TEST_LibraryTasksPolled(unsigned int testIndex, SYS_MODULE_INDEX libraryIndex,
SYS_MODULE_TASKS_ROUTINE libraryTasks);
```

Returns

None.

Description

This routine calls a tasks function of a library under test, indirectly under control of the test harness once the test is ready to verify the library.

Remarks

This routine must be called from the SYS_Tasks function or the appropriate RTOS thread loop when the library under test is configured for polled mode.

This routine will prevent the library's tasks function from being called before the library has been initialized.

Preconditions

The system, harness, and test must have been initialized.

Example

```
TEST_LibraryTasksPolled();
```

Parameters

Parameters	Description
testIndex	Index in test list to the test for this library.
libraryIndex	Index in test list for this instance of the library.
libraryTasks	Pointer to the library's Tasks function.

Function

void TEST_LibraryTasksPolled (unsigned int testIndex,
SYS_MODULE_INDEX libraryIndex,
SYS_MODULE_TASKS_ROUTINE libraryTasks)

TEST_LibraryTasksISR Function

Calls an ISR-based tasks function of the library under test.

File

test_harness.h

C

```
void TEST_LibraryTasksISR(unsigned int testIndex, SYS_MODULE_INDEX moduleIndex, SYS_MODULE_TASKS_ROUTINE
libraryTasks, INT_SOURCE source);
```

Returns

None.

Description

This routine calls an ISR-based tasks function of a library under test, indirectly under control of the test harness once the test is ready to verify the library.

Remarks

This routine must be called from the appropriate ISR vector function when the module under test is configured for interrupt mode.

Preconditions

The system, harness, and test must have been initialized.

Example

```
void __ISR(_TIMER_2_VECTOR, ipl2) _IntHandlerDrvTmrInstance1(void)
{
    TEST_LibraryTasksISR(2, 0, SAMPLE_Tasks, INT_SOURCE_TIMER_2);
}
```

Parameters

Parameters	Description
testIndex	Index to the APP_TH_TEST_DATA data structure passed to the test harness's "TEST_HarnessTasks" routine during system initialization.
moduleIndex	Index to the appropriate module instance.
libraryTasks	Pointer to the appropriate "Tasks" function for the library under test.
source	Interrupt source ID.

Function

void TEST_LibraryTasksISR (unsigned int testIndex,
SYS_MODULE_INDEX libraryTasks,
SYS_MODULE_TASKS_ROUTINE mutTasks,
INT_SOURCE source)

b) Platform Initialization Functions

TEST_PlatformInitialize Function

Called by the test harness to initialize configuration-specific test support.

File

test_harness.h

C

```
bool TEST_PlatformInitialize();
```

Returns

- true - If successfully able to initialize the configuration-specific test support.
- false - If not able to successfully initialize the configuration-specific test support.

Description

This function is called by the test harness to initialize configuration- specific test support.

Remarks

A weak implementation of this function is provided in the test harness library. If no external implementation is defined, then the internal weak implementation (which simply returns "true") will be used.

Preconditions

The system must have been initialized.

Example

None.

Function

bool TEST_PlatformInitialize (void)

TEST_PlatformReady Function

Indicates that the test configuration is ready and testing can begin.

File

test_harness.h

C

```
void TEST_PlatformReady();
```

Returns

None.

Description

This function indicates that the test configuration is ready and testing can begin. The purpose of this function is to allow the call to the TEST_PlatformInitialize function to return false when the configuration requires some time to complete its initialization.

Remarks

If TEST_PlatformInitialize returns "true", then it is not necessary to call this function.

Preconditions

The system, and test harness must have been initialized.

Example

TEST_PlatformReady();

Function

void TEST_PlatformReady (void)

c) Test Results Functions

TEST_HasCompleted Function

Allows a test to indicate that it has completed.

File

test_harness.h

C

```
void TEST_HasCompleted();
```

Returns

None.

Description

This function allows a test to indicate that it has completed. It is used to prematurely end the test.

Remarks

The harness will automatically end the test when the test's "Status" function indicates it has completed.

The test harness will accumulate and report subtest results using SYS_PRINT and/or SYS_MESSAGE macros. The test does not need to report results. Test only need to report debug info using SYS_DEBUG_* macros to help identify the reason for any failure.

A test is considered to have passed if all sub-tests pass. If any sub-tests fail, the test is considered to have failed. If no subtest results are reported, the test is considered to have failed.

Preconditions

The test harness must have been initialized.

Example

```
// Indicate that a test with one subtest has passed.
TEST_HasPassedSubtest(true);
TEST_HasCompleted();
```

Function

void TEST_HasCompleted (void)

TEST_HasPassedSubtest Function

Allows a test module to indicate that an individual subtest has completed and identifies if it has passed or failed.

File

test_harness.h

C

```
void TEST_HasPassedSubtest(bool passed);
```

Returns

None.

Description

This function allows a test module to indicate that an individual subtest has completed and identifies if it has passed or failed.

Remarks

The test harness will accumulate and report subtest results using SYS_PRINT and/or SYS_MESSAGE macros. The test module does not need to report results. Tests only need to report debug info using SYS_DEBUG_* macros to help identify the reason for any failure.

A test is considered to have passed if all sub-tests pass. If any sub-tests fail, the test is considered to have failed. If no subtest results are reported, the test is considered to have failed.

Preconditions

The application must have been initialized.

Example

```
// Indicate that the subtest has passed.
TEST_HasPassedSubtest(true);
```

Function

void TEST_HasPassedSubtest (bool passed)

TEST_LibraryInitialize Function

Indicates that the test is ready for the harness to initialize the library under test.

File

test_harness.h

C

```
bool TEST_LibraryInitialize();
```

Returns

- true If initialization of the library under test completed.
- false If initialization of the library under test has not yet completed.

Description

This function indicates that the test is ready for the harness to initialize the library under test.

Remarks

None.

Preconditions

The system, test harness, and test must have been initialized.

Example

if (TEST_LibraryInitialize()) { // begin testing }

Function

bool TEST_LibraryInitialize (void)

d) Test Time-out Functions

TEST_TimerIncrement Function

Increments test timeout timer.

File

test_harness.h

C

```
void TEST_TimerIncrement();
```

Returns

None.

Description

This function increments the test timeout counter by a number of milliseconds defined by TEST_TIMER_MS_PER_INCREMENT.

Remarks

This function should be called by the interrupt or alarm callback from the timer used by the test harness.

Preconditions

The system, test harness, test and timer must have been initialized.

Example

```
void __ISR(_TIMER_1_VECTOR, IPL1AUTO) _IntHandlerDrvTmrInstance0(void)
{
    TEST_TimerIncrement();
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
}
```

Function

void TEST_TimerIncrement (void)

e) Test Harness Initialization Data Types

LIBRARY_UNDER_TEST Structure

File

test_harness.h

C

```
typedef struct _library_under_test {
  SYS_MODULE_INITIALIZE_ROUTINE initialize;
  SYS_MODULE_REINITIALIZE_ROUTINE reinitialize;
  SYS_MODULE_DEINITIALIZE_ROUTINE deinitialize;
  SYS_MODULE_STATUS_ROUTINE status;
  unsigned int numberOfInstances;
  unsigned int numberOfTasks;
  SYS_MODULE_TASKS_ROUTINE tasks[TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY];
  SYS_MODULE_INIT * initData[TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY];
} LIBRARY_UNDER_TEST;
```

Members

Members	Description
SYS_MODULE_INITIALIZE_ROUTINE initialize;	Pointer to the library's initialization routine.
SYS_MODULE_REINITIALIZE_ROUTINE reinitialize;	Pointer to the library's reinitialization routine.
SYS_MODULE_DEINITIALIZE_ROUTINE deinitialize;	Pointer to the library's deinitialization routine.
SYS_MODULE_STATUS_ROUTINE status;	Pointer to the library's status routine.
unsigned int numberOfInstances;	Number of instances of the library under test.
unsigned int numberOfTasks;	Number of tasks routines required by each instance of the library under test.
SYS_MODULE_TASKS_ROUTINE tasks[TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY];	Array of pointers to the library's "Tasks" routine(s).
SYS_MODULE_INIT * initData[TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY];	Array of pointers to the library's init data structures.

Description

This is type LIBRARY_UNDER_TEST.

TEST_DATA Structure

Holds data necessary to initialize and run a single test.

File

test_harness.h

C

```
typedef struct _test_data {
  char * name;
  TEST_MODULE_INITIALIZE_ROUTINE initialize;
  TEST_MODULE_TASKS_ROUTINE tasks[TEST_HARNESS_MAX_NUM_TASKS];
  TEST_MODULE_START_FUNCTION start;
  TEST_MODULE_STATUS_ROUTINE status;
  SYS_MODULE_INDEX index;
  SYS_MODULE_INIT * initData;
  LIBRARY_UNDER_TEST * library;
  SYS_MODULE_OBJ testObj;
  unsigned int passed;
  unsigned int count;
  bool completed;
```

```
} TEST_DATA;
```

Members

Members	Description
char * name;	Pointer to test name string.
TEST_MODULE_INITIALIZE_ROUTINE initialize;	Pointer to test's initialize routine.
TEST_MODULE_TASKS_ROUTINE tasks[TEST_HARNESS_MAX_NUM_TASKS];	Pointer to test's tasks routine.
TEST_MODULE_START_FUNCTION start;	Pointer to test's start routine.
TEST_MODULE_STATUS_ROUTINE status;	Pointer to test's status routine.
SYS_MODULE_INDEX index;	Test module's index.
SYS_MODULE_INIT * initData;	Initial data for test module.
LIBRARY_UNDER_TEST * library;	Pointer to library under test data structure.
SYS_MODULE_OBJ testObj;	Test module's object handle.
unsigned int passed;	Count of sub-tests that have passed.
unsigned int count;	Count of sub-tests that have been run.
bool completed;	Flag indicating all sub-tests completed (module is idle).

Description

Test Data

This structure holds the data necessary to initialize and run a single test.

TEST_INIT_DATA Structure

Holds the test harness's initialization data.

File

test_harness.h

C

```
typedef struct _test_init_data {
  unsigned int numberOfTests;
  TEST_DATA * tests;
} TEST_INIT_DATA;
```

Members

Members	Description
unsigned int numberOfTests;	Number of tests.
TEST_DATA * tests;	Pointer to test data array.

Description

Test Harness Initialization Data

This structure holds the test harness's initialization data.

f) Test Interface Function Pointer Types

TEST_MODULE_INITIALIZE_ROUTINE Type

Pointer to a routine that initializes a test module.

File

test_harness.h

C

```
typedef SYS_MODULE_INITIALIZE_ROUTINE TEST_MODULE_INITIALIZE_ROUTINE;
```

Returns

A handle to the instance of the test module that was initialized. This handle is a necessary parameter to all of the other harness level routines for this test.

Description

This data type is a pointer to a routine that initializes a test module.

Remarks

This function will only be called once per test instance when the harness is ready to initialize the test instance.

Preconditions

The system and harness initialization must have been completed before the harness state machine will call the initialization routine for any tests.

Parameters

Parameters	Description
index	Identifier for the test instance to be initialized
init	Pointer to the data structure containing any data necessary to initialize the test. This pointer may be null if no data is required and default initialization is to be used.

Function

SYS_MODULE_OBJ TEST__Initialize) (

const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init)

TEST_MODULE_START_FUNCTION Type

Pointer to a routine that starts the test module.

File

test_harness.h

C

```
typedef bool (* TEST_MODULE_START_FUNCTION)(SYS_MODULE_OBJ object);
```

Returns

- true - If the test module is successfully able to start testing.
- false - If the test module is unable to start testing.

Description

This is a pointer to a routine that starts the test application.

Remarks

Each test module's "Start" function is called via a pointer to begin running the test contained in that module.

Preconditions

The test harness application and the test application must have been initialized and "running" (i.e., the test applications tasks function must be called from the test harness's state machine).

Example

None.

Parameters

Parameters	Description
object	Object handle to the module instance

Function

bool TEST__Start (SYS_MODULE_INDEX index)

TEST_MODULE_STATUS_ROUTINE Type

Pointer to a routine that gets the current status of a test module.

File

test_harness.h

C

```
typedef SYS_MODULE_STATUS_ROUTINE TEST_MODULE_STATUS_ROUTINE;
```

Returns

One of the possible status codes from SYS_STATUS, extended to include:

SYS_STATUS_TEST_COMPLETED - Indicates the test has completed.

Description

This data type is a pointer to a routine that gets the current status of a test module.

Remarks

A test's status operation can be used to determine when the test is busy or has completed as well as to obtain general status of the test.

If the status operation returns SYS_STATUS_BUSY, the previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). A module may define module-specific error values of less or equal SYS_STATUS_ERROR_EXTENDED (-10).

The status function must NEVER block.

If the status operation returns an error value the test has failed.

Preconditions

The system and harness initialization must have (and will be) completed and the test module's initialization routine will have been called before the harness will call the status routine for a test.

Example

None.

Parameters

Parameters	Description
object	Handle to the module instance

Function

SYS_STATUS TEST__Status (SYS_MODULE_OBJ object)

TEST_MODULE_TASKS_ROUTINE Type

Pointer to a routine that performs the tasks necessary to maintain the state machine of a test module system module.

File

test_harness.h

C

```
typedef void (* TEST_MODULE_TASKS_ROUTINE)(SYS_MODULE_OBJ object, unsigned int index);
```

Returns

None.

Description

This data type is a pointer to a routine that performs the tasks necessary to maintain the state machine of a test module module.

Remarks

A test module can have one or more (up to TEST_HARNESS_MAX_NUM_TASKS) tasks functions.

Preconditions

The system and harness initialization must have been completed and the test's initialization routine called before the harness will call any of the test's tasks routines.

Example

None.

Parameters

Parameters	Description
object	Handle to the test instance object
index	Index identifying which tasks function is being called. This index can be used to identify if a single task function is called from a different tasks context.

Function

void TEST__Tasks (SYS_MODULE_OBJ object, unsigned int index)

Files

Files

Name	Description
test_harness.h	This header file provides prototypes and definitions for the MPLAB Harmony test harness.
test_config_template.h	Test harness configuration template file.

Description

This section lists the source and header files used by the TEST HARNESS LIB Library.

test_harness.h

This header file provides prototypes and definitions for the MPLAB Harmony test harness.

Functions

	Name	Description
	TEST_HarnessTasks	MPLAB Harmony test harness' tasks function.
	TEST_HasCompleted	Allows a test to indicate that it has completed.
	TEST_HasPassedSubtest	Allows a test module to indicate that an individual subtest has completed and identifies if it has passed or failed.
	TEST_Initialize	MPLAB Harmony test harness initialization routine.
	TEST_LibraryInitialize	Indicates that the test is ready for the harness to initialize the library under test.
	TEST_LibraryTasksISR	Calls an ISR-based tasks function of the library under test.
	TEST_LibraryTasksPolled	Calls the a tasks function of a library under test when the test harness is ready to run the library.
	TEST_PlatformInitialize	Called by the test harness to initialize configuration-specific test support.
	TEST_PlatformReady	Indicates that the test configuration is ready and testing can begin.
	TEST_Tasks	Runs the current test's tasks functions.
	TEST_TimerIncrement	Increments test timeout timer.

Macros

	Name	Description
	TEST_HARNESS_MAX_NUM_INSTANCES_PER_LIBRARY	Defines the maximum number of instances of a single library under test.

Structures

	Name	Description
	_library_under_test	This is type LIBRARY_UNDER_TEST.
	_test_data	Holds data necessary to initialize and run a single test.
	_test_init_data	Holds the test harness's initialization data.
	LIBRARY_UNDER_TEST	This is type LIBRARY_UNDER_TEST.
	TEST_DATA	Holds data necessary to initialize and run a single test.
	TEST_INIT_DATA	Holds the test harness's initialization data.

Types

	Name	Description
	TEST_MODULE_INITIALIZE_ROUTINE	Pointer to a routine that initializes a test module.
	TEST_MODULE_START_FUNCTION	Pointer to a routine that starts the test module.
	TEST_MODULE_STATUS_ROUTINE	Pointer to a routine that gets the current status of a test module.
	TEST_MODULE_TASKS_ROUTINE	Pointer to a routine that performs the tasks necessary to maintain the state machine of a test module system module.

Description

MPLAB Harmony Test Harness Header File

This header file provides function prototypes and data type definitions for the test harness interface.

File Name

test_harness.h

Company

Microchip Technology Inc.

test_config_template.h

Test harness configuration template file.

Macros

	Name	Description
	TEST_HARNESS_MAX_NUM_TASKS_PER_LIBRARY	Defines the maximum number of "Tasks" routines implemented by a single library under test.
	TEST_IDLE_SLEEP_MS	Defines the number of milliseconds the test harness will sleep when the test reports that it is idle.
	TEST_IDLE_SLEEP_MS_LIBRARY	Defines the number of milliseconds the library tasks function will sleep when the library is idle.
	TEST_TIMER_CLOCK_PRESCALER	Defines the selected clock prescaler value for the timeout timer.
	TEST_TIMER_CLOCK_SOURCE	Defines the selected clock source for the timer used for the timeout timer.
	TEST_TIMER_ID	Defines the timer peripheral used for the test timeout timer.
	TEST_TIMER_INCREMENT_PERIOD	Defines the number of timer counts that make up a single timer period.
	TEST_TIMER_INTERRUPT_PRIORITY	Defines the interrupt priority for the selected test timer.
	TEST_TIMER_INTERRUPT_SOURCE	Defines the interrupt source ID of the timer used for the timeout timer.
	TEST_TIMER_INTERRUPT_SUBPRIORITY	Defines the interrupt subpriority for the selected test timer.
	TEST_TIMER_INTERRUPT_VECTOR	Defines the interrupt vector for the selected test timer.
	TEST_TIMER_MS_PER_INCREMENT	Defines the number of milliseconds per timer increment period.
	TEST_TIMER_MS_TIMEOUT	Defines the number of milliseconds that must expire before a test will time out.

Description

Test Harness Configuration Template

This file provides the list of all the configurations that can be used with the test harness. This file should not be included by any source files. It is strictly for documentation.

File Name

test_config_template.h

Company

Microchip Technology Inc.

Index