Trampoline (OSEK/VDX OS) Test Implementation - Version 1.0

Florent PAVIN ; Jean-Luc BECHENNEC

December 3, 2009

Contents

1 Introduction

This document describes the test implementation of the Trampoline test procedure [2]. It is based on *Guide de mise en oeuvre des tests de non régression* [1].

2 Tests implementation

To implement the tests sequences of the Trampoline test procedure [3], Embedded Unit [4] is used.
The first section is thus what is Embedded Unit, how to use it to check if a variable has the right value, and what we add to the software to check the scheduling of our tests.
The second section will define the test code organisation and distribution with a test

sequence example.
To conclue the tests implementation, a configuration file has to be created in order to garantee the portage of the tests from every operating system (Unix, Windows...).

2.1 Embedded Unit

2.1.1 Embedded Unit explanation

To check a test sequence with Embedded Unit, three functions has to be called. The first is to start a new Embedded Unit check with *TestRunner_start()*, the second is to call the test sequence by *TestRunner_runTest(function_to_call())* and the last one is to finish Embedded Unit check and display information about the tests which have just been tested with *TestRunner_end()*.
You should create *function_to_call()* as below to respect EmbUnit syntax :

```
/*create the test suite with all the test cases*/
TestRef function_to_call(void)
{
    EMB_UNIT_TESTFIXTURES(fixtures) {
        new_TestFixture("Instance_Name1_of_task1",Instance_Name1_of_task1)
        new_TestFixture("Instance_Name2_of_task1",Instance_Name2_of_task1)
    };
    EMB_UNIT_TESTCALLER(Test_Sequence_Name,"Test_Sequence_Name",NULL,NULL
        ,fixtures);

    return (TestRef)&Test_Sequence_Name;
}
```

You can create as many "new_TestFixture" as you want, with differents *Instance_Names()* and make your tests with TEST_ASSERT_EQUAL_INT macro as shown below :

```
/*test case:test the reaction of the system called with an activation of
    a task*/
static void Instance_Name1_of_task1(void)
{
    int result_inst_1, result_inst_2;

    result_inst_1 = ActivateTask(t0);
    TEST_ASSERT_EQUAL_INT(E_OS_ID , result_inst_1);

    result_inst_2 = ActivateTask(t2);
    TEST_ASSERT_EQUAL_INT(E_OK , result_inst_2);
}
```

You can call different *TestRunner_runTest(function_to_call())* with different *function_to_call*.
If all your tests are good, Embedded Unit should display

```
(OK n tests)
```

n is *TestRunner_runTest()* call number.
If one of your test is wrong, Embedded Unit should display

```
([Test_Sequence_Name].[Instance_Name1_of_task1] ([file].c [line]) exp [value_expected]
was [value_was])
run n failure x
```

telling you exactly which is the wrong TEST_ASSERT_EQUAL_INT and what the value should be equal to ([value_expected] instead of [value_was]).
The number of failed tests (x) and the total sequence number (n) are displayed too.

2.1.2 Adds to Embedded Unit

As Embedded Unit can't tell the user about the scheduling of a test sequence, we decided to add an upstrem macro to TEST_ASSERT_EQUAL_TIME.
In order to check scheduling, we decided to create a global variable. Before each service call, a Macro is called to check if the service number (in parameter), is equal to the global variable incremented. After each service call, an other Macro is called to check if the service number (in parameter), is equal to the global variable. The second Macro is here to check a switch context and the first Macro to check the creation of a new task. The two macros code are (from *SchedulingCheck.h*) :

```
#define SCHEDULING_CHECK_INIT(number)                                          \
{                                                                              \
   extern unsigned char test_number;                                           \
   if ((number) != (++test_number)){                                           \
      assertImplementationInt((number),(test_number),__LINE__,__FILE__);\
   }
#define SCHEDULING_CHECK_AND_EQUAL_INT(number, expected, actual)               \
   if ((number) == (test_number)){                                             \
      TEST_ASSERT_EQUAL_INT((expected),(actual))                               \
   }else{                                                                      \
      assertImplementationInt((number),(test_number),__LINE__,__FILE__);\
   }                                                                           \
}
```

test_number is previously declared and set to zero (see *Scheduling.c*). If sereval tests are needed, SCHEDULING_CHECK_AND_EQUAL_INT_FIRST (see below) is used to check the first variable(s) but the last macro has to be SCHEDULING_CHECK_AND_EQUAL_INT to close the "bracket".

```
#define SCHEDULING_CHECK_AND_EQUAL_INT_FIRST(number, expected, actual)         \
   if ((number) == (test_number)){                                             \
      TEST_ASSERT_EQUAL_INT((expected),(actual))                               \
   }else{                                                                      \
      assertImplementationInt((number),(test_number),__LINE__,__FILE__);\
   }
```

An other macro is created to check the scheduling without testing a service call, it is SCHEDULING_CHECK_STEP :

```
#define SCHEDULING_CHECK_STEP(number)                                          \
{                                                                              \
   extern unsigned char test_number;                                           \
   if ((number) != (++test_number)){                                           \
      assertImplementationInt((number),(test_number),__LINE__,__FILE__);\
   }                                                                           \
}
```

Instance_Name1_of_task1() became :

```
/*test case:test the reaction of the system called with an activation of
     a task*/
static void Instance_Name1_of_task1(void)
{
   int result_inst_1, result_inst_2;

   SCHEDULING_CHECK_INIT(1);
   result_inst_1 = ActivateTask(t0);
   SCHEDULING_CHECK_AND_EQUAL_INT(1 , E_OS_ID , result_inst_1);

   SCHEDULING_CHECK_INIT(2);
   result_inst_2 = ActivateTask(t2);
   SCHEDULING_CHECK_AND_EQUAL_INT(3 , E_OK , result_inst_2);

   SCHEDULING_CHECK_STEP(4);
}
```

with

```
/*test case:test the reaction of the system called with an activation of
     a task*/
static void Instance_Name1_of_task2(void)
{
   int result_inst_1;

   SCHEDULING_CHECK_INIT(3);
   result_inst_1 = TerminateTask();
   SCHEDULING_CHECK_AND_EQUAL_INT(3 , E_OK , result_inst_1);
}
```

As task2 has a priority higher than task1, its activation in task1 should preempt task1 (if the mode is preemptive). Scheduling tests have to check that task2 is executed after ActivateTask(t2). The returned value of TerminateTask() can't be tested if the service goes well, it is just here in case the service doesn't go properly. In *Instance_Name1_of_task1*, SCHEDULING_CHECK_STEP(4) is not useful because the previous macro check that task2 has preempted task1, but it is here to show how to use it.

2.2 Test code organisation and distribution

As EmbUnit is described above, this section explains functions (see *config.c*) we have created to make tests easier to understand and implement with a test sequence example in Annexe A.

2.2.1 Send Interrupts

We have created a function to send interrupt to Trampoline. As this function is not portable, it is explained later in 2.3.

2.2.2 Alarm Test Sequences

As we discovered that the computer runs the program too fast and the alarm expiration is hardware dependant, we decided to create a synchronization between the task which creates the alarm and the activation of this alarm. We did that in two different maner,

depending on a periodic alarm or a one shot alarm.
Indeed, in order to wait the activation of a periodic alarm we check the number of ticks left before this activation, save this value, and when the number of ticks is higher than the old one, it means the counter restart from "cycle" and the alarm has been activated. We have to fix "cycle" at more than '1' otherwise it's always '1' and we don't know when the alarm has been activated. We fix '2' in every test sequences. We have then to "force scheduling".
If it's a one shot alarm, a call to *SetRelAlarm()* returns E_OS_STATE because the alarm is already active, but when it returns E_OK, it means the previous alarm has been activated. We have to cancel the last alarm and "force scheduling".
Those functions should be called from every test sequences thus, they are in an uppstream file (see *./config.c*). The functions are below :

```
void WaitActivationPeriodicAlarm(AlarmType Alarm){
    u32 temp, result_inst_;
    TickType result_inst_tt;
    result_inst_tt = 0;
    do{
        temp = result_inst_tt;
        result_inst_ = GetAlarm(Alarm,&result_inst_tt);
    }while( (temp >= result_inst_tt) || (temp==(0)) );
}

void WaitActivationOneShotAlarm(AlarmType Alarm){
    int result_inst_;
    TickType result_inst_tt;
    do{
        GetAlarm(Alarm,&result_inst_tt);
    }while((SetRelAlarm(Alarm, 1, 0) == E_OS_STATE));

    result_inst_ = CancelAlarm(Alarm);
    TEST_ASSERT_EQUAL_INT(E_OK, result_inst_);

}
```

The incrementation of the counter on Unix System is a random bit. Sometimes, it is incremented three or four times in a row so in **preemptive** mode, if TICKPERBASE is equal to '1', the alarm is activated several times in a row and the number of ticks can always be '2' (if TICKPERBASE=2 in the OIL file) and never '1'. The activation of the alarm is thus not detected by the function and the test sequence goes wrong. To correct this problem, we fix TICKPERBASE to '10' in every test sequences.

2.3 Tests portage : configuration file - IN PROGRESS

As we said above, we have to be careful not to create OS dependent function to ensure the tests portage. Like interrupts functions which is different weather the tests are executed in Unix, Window... operating system. This section goes all over the portable functions.

2.3.1 Interrupts trigger

On Unix systems, interrupts are trigged using the *kill* function and can be done as showed below :

```
void tpl_send_it1(void){
    int ipid;
    ipid = getpid();
    kill(ipid,SIGUSR1);
}
```

On Windows systems (use different fonction or a software to translate Unix functions on Windows ones ?) :

```
    windows function..
```

2.3.2 Embedded Unit output

As we said earlier, Embedded Unit displays on the screen weather the tests are good or wrong. *embUnit/config.h* contains a macro which calls the display function according to the machine :

- *printf* on Unix systems
- *windows print* on Windows systems

3 Tests automatisation...

3.1 ...from 'functional' directory

In order to make all the tests and to be able to tell to the user if they've been executed correctly, we have created a shell script. This shell script build and execute all the test sequences and compare the results (saved in a file) to the expected ones (an other file). The shell script is below. The names of the test sequences are in 'testSequences.txt'. The "clean" case deletes all the files that have been created to generated the executable file and the embUnit library (because it is OS dependent).

```
#!/bin/sh

######
# performs all the functional tests.
# args:
# (nothing): performs tests
# clean    : call make clean for each tests folders
# {goil's target} : if Makefile hasn't been created yet, a 'goil' command
    is needed with the target name (default=libpcl)
#
# TODO :
#      delete testSequences file and do the loops for each directory (which
    contains defaultAppWorkstation.oil) #ls -d
######

#check target
parameters=false
while [ $# -gt 0 ]
do
    #echo "param tres : $1"
```

```
    if [ "`echo $1 | grep -c =`" = "1" ]
    then
        #echo "param tre with ="
        option=`echo $1 | awk -F"=" '{print $1}'`
        #echo "option : $option"
        case "$option" in
            -t) target=`echo $1 | awk -F"=" '{print $2}'`;
                echo "Target $target selected";;
        esac
    else
        case "$1" in
            clean) target="clean";;
            *) echo "Wrong parameters!";
               echo "Waited :";
               echo "    -t=[target],";
               echo "    clean or";
               echo "    NULL (posix target selected)"; exit 1;;
        esac
    fi
    parameters=true
     shift
done
if [ "$parameters" = "false" ]
then
    #default target = posix
    target="posix"
    echo "Target $target selected"
fi


#GOIL tests
cd ./GOIL
./GOIL_tests.sh $target

#functional tests
cd ../functional
./functional_tests.sh $target


cd ..

if [ "$target" != "clean" ]
then
    #Compare results
    echo "Compare results with the expected ones..."
    if [ `diff functional/functional_results_expected.log functional/
        functional_results.log | wc -l` -eq 0 ]
    then
        echo "Functional tests Succeed!!"
    else
        echo "Functional tests Failed! Results are stored in `pwd`/functional
            /functional_results.log"
    fi
    if [ `diff GOIL/GOIL_results_expected.log GOIL/GOIL_results.log | wc -l`
         -eq 0 ]
    then
        echo "GOIL tests Succeed!!"
    else
```

```
        echo "GOIL tests Failed! Results are stored in `pwd`/GOIL/
            GOIL_results.log"
    fi
fi
```

A Test sequence example : alarms_s11

```
/**
 * @file alarms_s11.oil
 *
 * @section desc File description
 *
 * @section copyright Copyright
 *
 * Trampoline Test Suite
 *
 * Trampoline Test Suite is copyright (c) IRCCyN 2005-2007
 * Trampoline Test Suite is protected by the French intellectual property
     law.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; version 2
 * of the License.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
     02110-1301, USA.
 *
 * @section infos File informations
 *
 * $Date$
 * $Rev$
 * $Author$
 * $URL$
 */

OIL_VERSION = "2.5" : "alarms_s11" ;

IMPLEMENTATION trampoline {

    TASK {
        UINT32 STACKSIZE = 32768 ;
        UINT32 [1..10] PRIORITY = 1 ;
    } ;
};

 CPU test {
```

```
 OS config {
      STATUS = STANDARD;
    APP_SRC = "alarms_s11.c";
    APP_SRC = "task1_instance.c";
    APP_SRC = "task2_instance.c";
    APP_SRC = "task3_instance.c";
    APP_SRC = "../config.c";
      TRAMPOLINE_BASE_PATH = "../../..";
      APP_NAME = "alarms_s11_exe";
    CFLAGS = "-I../../embUnit";
      CFLAGS  = "-Werror -Wall -pedantic";
      CFLAGS  = "-Wmissing-field-initializers";
    LDFLAGS = "-L../../lib -lembUnit";
   SHUTDOWNHOOK = TRUE;
 } ;

 APPMODE std {};

TASK t1 {
    AUTOSTART = TRUE { APPMODE = std ; } ;
    PRIORITY = 1;
    ACTIVATION = 1;
    SCHEDULE = FULL;
};
TASK t2 {
    AUTOSTART = FALSE ;
    PRIORITY = 2;
    ACTIVATION = 1;
    SCHEDULE = FULL;
};
TASK t3 {
    AUTOSTART = FALSE ;
    PRIORITY = 2;
    ACTIVATION = 1;
    SCHEDULE = FULL;
};
COUNTER Counter1{
    MAXALLOWEDVALUE = 15;
    TICKSPERBASE = 10;
    MINCYCLE = 1;
};
ALARM Alarm1{
    COUNTER = Counter1;
    ACTION = ACTIVATETASK {
        TASK = t2;
    };
      AUTOSTART = TRUE {
           ALARMTIME = 7;
           CYCLETIME = 0;
           APPMODE = std;
    };
};
ALARM Alarm2{
    COUNTER = Counter1;
    ACTION = ACTIVATETASK {
        TASK = t3;
    };
      AUTOSTART = TRUE {
```

```
            ALARMTIME = 15;
            CYCLETIME = 15;
            APPMODE = std;
        };
    };
};

/* End of file alarms_s11.oil */
```

```
/**
 * @file alarms_s11/alarms_s11.c
 *
 * @section desc File description
 *
 * @section copyright Copyright
 *
 * Trampoline Test Suite
 *
 * Trampoline Test Suite is copyright (c) IRCCyN 2005-2007
 * Trampoline Test Suite is protected by the French intellectual property
     law.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; version 2
 * of the License.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
     02110-1301, USA.
 *
 * @section infos File informations
 *
 * $Date$
 * $Rev$
 * $Author$
 * $URL$
 */

#include "tpl_os.h"
#include "embUnit.h"

TestRef ALARMSTest_seq11_t1_instance(void);
TestRef ALARMSTest_seq11_t2_instance(void);
TestRef ALARMSTest_seq11_t3_instance(void);

StatusType instance_t3 = 0;

int main(void)
{
    StartOS(OSDEFAULTAPPMODE);
    return 0;
```

```
}

void ShutdownHook(StatusType error)
{
    TestRunner_end();
}

TASK(t1)
{
    TestRunner_start();
    TestRunner_runTest(ALARMSTest_seq11_t1_instance());
    ShutdownOS(E_OK);
}

TASK(t2)
{
    TestRunner_runTest(ALARMSTest_seq11_t2_instance());
}

TASK(t3)
{
    TestRunner_runTest(ALARMSTest_seq11_t3_instance());
}

/* End of file alarms_s11/alarms_s11.c */
```

```
/**
 * @file alarms_s11/task1_instance.c
 *
 * @section desc File description
 *
 * @section copyright Copyright
 *
 * Trampoline Test Suite
 *
 * Trampoline Test Suite is copyright (c) IRCCyN 2005-2007
 * Trampoline Test Suite is protected by the French intellectual property
     law.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; version 2
 * of the License.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
     02110-1301, USA.
 *
 * @section infos File informations
 *
 * $Date$
 * $Rev$
```

```
 * $Author$
 * $URL$
 */

/*Instance of task t1*/

#include "embUnit.h"
#include "tpl_os.h"

DeclareAlarm(Alarm1);
DeclareAlarm(Alarm2);

void WaitActivationOneShotAlarm(AlarmType Alarm);
void WaitActivationPeriodicAlarm(AlarmType Alarm);

/*test case:test the reaction of the system called with
an activation of a task*/
static void test_t1_instance(void)
{
    StatusType result_inst_1;

    SCHEDULING_CHECK_STEP(1);

    WaitActivationOneShotAlarm(Alarm1);

    SCHEDULING_CHECK_STEP(3);

    WaitActivationPeriodicAlarm(Alarm2);

    SCHEDULING_CHECK_INIT(5);
    result_inst_1 = CancelAlarm(Alarm2);
    SCHEDULING_CHECK_AND_EQUAL_INT(5,E_OK, result_inst_1);

    SCHEDULING_CHECK_STEP(6);


}

/*create the test suite with all the test cases*/
TestRef ALARMSTest_seq11_t1_instance(void)
{
    EMB_UNIT_TESTFIXTURES(fixtures) {
        new_TestFixture("test_t1_instance",test_t1_instance)
    };
    EMB_UNIT_TESTCALLER(ALARMSTest,"ALARMSTest_sequence11",NULL,NULL,
         fixtures);

    return (TestRef)&ALARMSTest;
}

/* End of file alarms_s11/task1_instance.c */
```

```
/**
 * @file alarms_s11/task2_instance.c
 *
 * @section desc File description
 *
 * @section copyright Copyright
```

```
 *
 * Trampoline Test Suite
 *
 * Trampoline Test Suite is copyright (c) IRCCyN 2005-2007
 * Trampoline Test Suite is protected by the French intellectual property
     law.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; version 2
 * of the License.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
     02110-1301, USA.
 *
 * @section infos File informations
 *
 * $Date$
 * $Rev$
 * $Author$
 * $URL$
 */

/*Instance of task t2*/

#include "embUnit.h"
#include "tpl_os.h"

/*test case:test the reaction of the system called with
an activation of a task*/
static void test_t2_instance(void)
{

   StatusType result_inst_1;

   SCHEDULING_CHECK_INIT(2);
   result_inst_1 = TerminateTask();
   SCHEDULING_CHECK_AND_EQUAL_INT(2,E_OK, result_inst_1);

}

/*create the test suite with all the test cases*/
TestRef ALARMSTest_seq11_t2_instance(void)
{
   EMB_UNIT_TESTFIXTURES(fixtures) {
      new_TestFixture("test_t2_instance",test_t2_instance)
   };
   EMB_UNIT_TESTCALLER(ALARMSTest,"ALARMSTest_sequence11",NULL,NULL,
       fixtures);

   return (TestRef)&ALARMSTest;
```

```
}

/* End of file alarms_s11/task2_instance.c */
```

```
/**
 * @file alarms_s11/task3_instance.c
 *
 * @section desc File description
 *
 * @section copyright Copyright
 *
 * Trampoline Test Suite
 *
 * Trampoline Test Suite is copyright (c) IRCCyN 2005-2007
 * Trampoline Test Suite is protected by the French intellectual property
     law.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; version 2
 * of the License.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
     02110-1301, USA.
 *
 * @section infos File informations
 *
 * $Date$
 * $Rev$
 * $Author$
 * $URL$
 */

/*Instance of task t3*/

#include "embUnit.h"
#include "tpl_os.h"

/*test case:test the reaction of the system called with
an activation of a task*/
static void test_t3_instance(void)
{

   StatusType result_inst_1;

   SCHEDULING_CHECK_INIT(4);
   result_inst_1 = TerminateTask();
   SCHEDULING_CHECK_AND_EQUAL_INT(4,E_OK, result_inst_1);

}
```

```
/*create the test suite with all the test cases*/
TestRef ALARMSTest_seq11_t3_instance(void)
{
    EMB_UNIT_TESTFIXTURES(fixtures) {
        new_TestFixture("test_t3_instance",test_t3_instance)
    };
    EMB_UNIT_TESTCALLER(ALARMSTest,"ALARMSTest_sequence11",NULL,NULL,
        fixtures);

    return (TestRef)&ALARMSTest;
}

/* End of file alarms_s11/task3_instance.c */
```

References

[1] Guillaume N. and Jonathan I. Guide de mise en oeuvre des tests de non-régression. Master's thesis, IUT Angers, June 2008.

[2] Florent PAVIN and Jean-Luc BECHENNEC. *Trampoline (OSEK/VDX) Test Plan.* IRCCYN, 1.0 edition, February 2009.

[3] Florent PAVIN and Jean-Luc BECHENNEC. *Trampoline (OSEK/VDX) Test Procedure.* IRCCYN, 1.0 edition, February 2009.

[4] Embedded Unit Project. *EmbUnit.* http://embunit.sourceforge.net/, 2003.