**The Report Committee for Fredrick Dean Mainor**
**Certifies that this is the approved version of the following report:**

# Using KLEE to Generate Test Cases for the Texas Instruments® Stellaris® Peripheral Driver Library

**APPROVED BY**
**SUPERVISING COMMITTEE:**

**Supervisor:**

Sarfraz Khurshid

William Bard

# Using KLEE to Generate Test Cases for the Texas Instruments® Stellaris® Peripheral Driver Library

**by**

**Fredrick Dean Mainor, B.S.E.E.**

**Report**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Master of Science in Engineering**

**The University of Texas at Austin**

**August 2014**

# Abstract


## Using KLEE to Generate Test Cases for the Texas Instruments® Stellaris® Peripheral Driver Library

Fredrick Dean Mainor, MSE

The University of Texas at Austin, 2014

Supervisor: Sarfraz Khurshid

Software engineers spend much of their time checking the correctness of software. Software testing is the most widely used technique for accomplishing this task. Most of the test cases used for checking software are manually created, and may not always cover all execution paths of the software. If key test cases are not executed, then the possibility of errors within the software still exists. By using tools that can automate the testing of software, software engineers can run exhaustive tests on their applications to provide verification and validation. Symbolic execution is a program analysis technique that can be utilized to achieve this. KLEE is an open-source dynamic test generation tool based on symbolic execution. In this report I present my results from evaluating KLEE on the Texas Instruments® Stellaris® Peripheral Driver Library. The Stellaris® Peripheral Driver Library consists of software drivers for controlling the peripherals on the Stellaris suite of ARM® Cortex-M based microcontrollers. In total 554 functions within the library were tested, and a total of 14763 test cases were generated.

There were 32 bugs found in the software, which include assertion violations, memory errors, and arithmetic errors (division by zero, and shift errors).

# Table of Contents

# List of Tables

# List of Figures

# 1. INTRODUCTION

Software engineers spend a lot of their time checking the correctness of software. It is crucial that the software be correct, especially when peoples' lives are at stake. Many software engineers perform a static quality analysis (QA) of their software by reviewing the source code. This technique is useful, but can be very tedious and may lead to bugs being overlooked if the code is complex and lengthy. Another technique that is more commonly used is executing specific modules in the software to ensure that it is executing as expected according to test cases that were prepared prior to executing the software. The problem with this technique arises when the software engineer has to manually generate the test cases. If key test cases are not executed, then the possibility of errors within the software still exists. By using tools that can automate the testing of software, software engineers can run exhaustive tests on their applications to provide verification and validation. Symbolic execution is a program analysis technique that can be utilized to achieve this. KLEE is an open-source test generation tool based on symbolic execution. In this report I present my results from evaluating KLEE on the Texas Instruments® Stellaris® Peripheral Driver Library (revision 10636). The Stellaris® Peripheral Driver Library consists of software drivers for controlling the peripherals on the Stellaris suite of ARM® Cortex-M based microcontrollers. KLEE was used to generate the test cases for each driver in the library, with the exception of the CPU driver. The next section describes symbolic execution. The third section discusses the design and functionality of KLEE. In the fourth section I discuss the Stellaris Peripheral Driver Library. The implementation of KLEE on the Stellaris Peripheral Driver Library is described in the fifth section. In the sixth section I present my evaluation results and then conclude. In total 554 functions within the library were tested,

and a total of 14763 test cases were generated. There were 32 bugs found in the software, which include assertion violations, memory errors, and arithmetic errors (division by zero, and shift errors).

## 2. SYMBOLIC EXECUTION

Symbolic execution is a program analysis technique that helps with the software validation/verification effort by generating software test cases automatically. The test cases that are generated cover many, if not all, execution paths of the application. Symbolic execution has been around since the 1970s, and was first used for simple applications with primitive data types, such as integers and booleans [1]. Instead of assigning program variables real values, symbolic execution uses symbols to represent the values. As the program is executed, the program variables that depend on symbolic inputs are updated to symbolic expressions. The idea behind using symbolic values instead of real values was to use the symbolic values to represent an infinite set of real values. Thus, the results of executing the program on one set of symbolic values would yield the same result if it were executed with an infinite number of real values [1]. When a conditional statement is evaluated the execution path condition is also updated. The path condition acquires and maintains all symbolic logic required to reach the current execution path. In order to fully cover all execution paths, when a conditional statement is evaluated the path condition is updated for all possible outcomes. This produces multiple execution paths resulting in branches in the symbolic execution tree. For example, if the current path condition was pc before a branch condition, and the conditional expression is q then (*pc & q*) will be the path condition for the true branch, and (*pc & !q*) will be the path condition for the false branch. When the execution terminates, the path condition is solved via a constraint solver (also referred to as a decision procedure) and the solution forms the test case. This allows for the substitution of real data into the program variables to follow the same execution path and terminate. An example of a symbolic execution tree, and the code used to construct the tree is

displayed in Figure 1 [7]. Another valuable feature of symbolic execution is that it is able to find run-time errors while it is traversing the symbolic execution tree. The major issue with traditional symbolic execution is how to deal with the exponential number of paths in the symbolic execution tree caused by loops and complex path conditions. This continues to be a problem for current implementations.

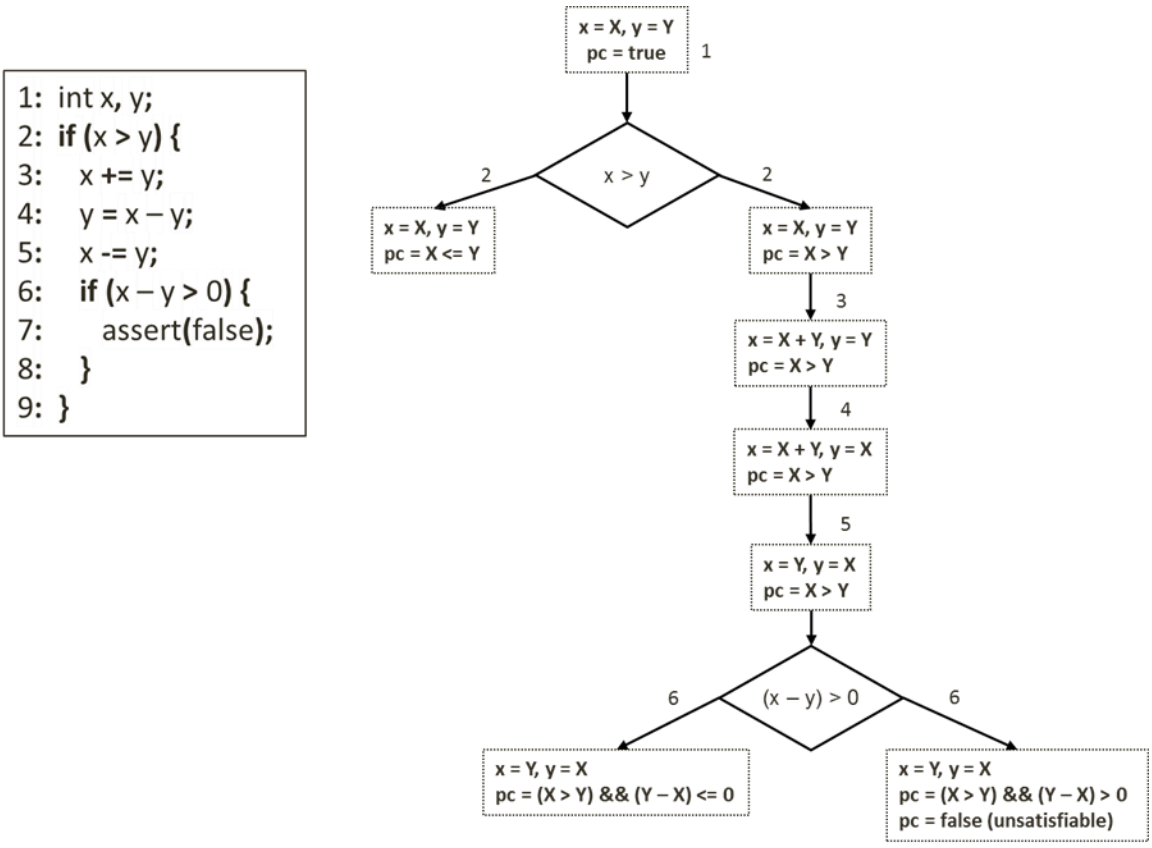


Figure 1. Symbolic execution tree constructed from code snippet.

## 3. KLEE

Dynamic test generation improves traditional symbolic execution by performing a mixture of concrete and symbolic execution at run-time. The applications are essentially executed without any changes to the source code. The statements that depend on symbolic input are treated differently and their constraints are added to the path condition [2]. KLEE was developed at Stanford in 2008 and is an extension of the dynamic test generation tool EXE [3]. KLEE performs a combination of concrete and symbolic execution on C applications to generate high line coverage test cases. KLEE works in conjunction with the publically available LLVM compiler for GNU C. An application's source code is first compiled into bitcode using the LLVM compiler. KLEE then performs symbolic execution on the LLVM bitcode. KLEE models memory with bit-level accuracy and uses heuristics to reduce the number of execution paths that need to be explored which results in higher line coverage. KLEE has two goals: (1) hit every line of executable code in the program and (2) detect at each dangerous operation (e.g., dereference, assertion) if any input value exists that could cause an error [3]. When the symbolic execution terminates, KLEE solves the current path condition using the STP constraint solver to produce a test case that will follow the same execution path when re-run on an unmodified version of the source code [3]. KLEE has proven to be a successful tool for automated software testing. KLEE was able to find 56 serious bugs when applied to several UNIX utility applications, some of which had been undetected for over 15 years [3]. Since KLEE is open source there is no cost involved in using the software tool for automated testing. The KLEE community offers valuable resources, such as online documentation, tutorials, mailing distributions, and many papers that use or extend KLEE [4].

# 4. STELLARIS® PERIPHERAL DRIVER LIBRARY

The Stellaris Peripheral Driver Library consists of twenty-three software drivers for controlling the peripherals on the Stellaris suite of ARM® Cortex-M based microcontrollers [5]. The driver library was developed by Texas Instruments and is written in the C programming language. Texas Instruments allows for royalty-free use of the software library provided that their copyright notice is retained in the source code [6]. Each peripheral and its software driver is listed Table 1. These drivers are referred to as the Software Driver Model of the Stellaris Peripheral Driver Library. Each model in the Stellaris suite of microcontrollers has a part-specific header file that defines all of the registers that are available for that microcontroller. By including this header file you can read/write to the peripheral registers directly. This direct access of the peripheral's registers is referred to as the Direct Register Access Model. This model allows the programmer to only access registers that are defined in the header file and eliminates the possibility of accessing peripherals that do not exist. The Direct Register Access Model should definitely be considered when memory or speed requirements are not being met. This report focuses on the drivers which make up the Software Driver Model of the Stellaris Peripheral Driver Library.

All of the functions defined in the CPU module contain inline assembly and are used primarily as instruction wrappers for special CPU instructions needed by the drivers. The driver library includes error handling to check for assertion violations of function arguments. The error handling is typically used only during the development stage and is removed once the product is released to allow for a smaller memory footprint and faster processing [5].

| | |
|---|---|
| Analog Comparator | comp.c |
| Analog to Digital Converter (ADC) | adc.c |
| Controller Area Network (CAN) | can.c |
| CPU | cpu.c |
| Ethernet Controller | ethernet.c |
| External Peripheral Interface (EPI) | epi.c |
| Flash | flash.c |
| GPIO | gpio.c |
| Hibernation Module | hibernate.c |
| Inter-Integrated Circuit (I2C) | i2c.c |
| Inter-IC Sound (I2S) | i2s.c |
| Interrupt Controller (NVIC) | interrupt.c |
| Memory Protection Unit (MPU) | mpu.c |
| Pulse Width Modulator (PWM) | pwm.c |
| Quadrature Encoder (QEI) | qei.c |
| Synchronous Serial Interface (SSI) | ssi.c |
| System Control | sysctl.c |
| System Tick (SysTick) | systick.c |
| Timer | timer.c |
| UART | uart.c |
| uDMA | udma.c |
| USB | usb.c |
| Watchdog Timer | watchdog.c |

Table 1. Stellaris suite peripherals and their software drivers.

There are several tool chains that are available to compile source code and load the binaries into the flash memory of the Stellaris microcontrollers. I used the Keil™ RealView® Microcontroller Development Kit (MDK) for development/testing on a Windows platform. The tool chain was used to confirm the errors that were reported by KLEE. The Stellaris EKK-LM3S8962 evaluation board was used to confirm the errors [10].

## 5. IMPLEMENTATION

To get started, I installed KLEE on a 64-bit CentOS 6.5 Linux distribution. KLEE performs symbolic execution on LLVM bitcode, so I installed the LLVM 2.9 compiler. The latest release of LLVM at the time of this report was LLVM 3.4.1. KLEE only provided experimental support for LLVM 3.4, but fully supported LLVM 2.9. KLEE also uses the STP constraint solver for solving path conditions. I installed STP rev 940, which is the tested and recommend version by the KLEE developers. The KLEE "Getting Started" page provides step-by-step instructions on building and running KLEE [4]. The latest release (10636) of the Stellaris Peripheral Driver Library was downloaded from the Texas Instruments website [6].

The latest release of the Stellaris Peripheral Driver Library contains 632 functions. This total includes both the static and deprecated functions as well. The functions that I am interested in evaluating are those that have one or more input parameters. There are 78 functions that have no input parameters (void). Thus, in total 554 functions were evaluated. One of the unique features of KLEE is that you can perform symbolic execution with little, if any, modifications to the source code under test. Many of the functions in the driver library use assertions to validate the input arguments. An example of this is shown below.

```
1: //
2: // Check the arguments.
3: //
4: ASSERT(ulBase == I2S0_BASE);
```

These assertions were replaced with KLEE assertions.

```
4: klee_assert(ulBase == I2S0_BASE);
```

As mentioned previously, static functions were evaluated as well, but in order to make the functions available in the KLEE function under test .c file, more on this later, I had to remove the static reference in the code. If the static keyword is not removed, then KLEE will report the error "external call with symbolic argument". There were two functions in the library that I had to add a break statement into a while loop to prevent it from spinning in an infinite loop. The EthernetPacketGet has a condition to wait for a packet to become available and the SysCtlPIOSCCalibrate has a condition to wait for a calibration to complete. The values from the registers that these conditions were waiting on were not treated as symbolic. I also had to remove an assertion based on the alignment of the RAM vector table in the IntRegister function. The RAM vector table alignment is set based on the compiler.

After these minor changes were made to the code under test, I needed to compile the code into LLVM bitcode with the command below

```
llvm-gcc -m32 --emit-llvm -I/root/KLEE/klee/include -
I/root/StellarisWare -L /root/KLEE/klee/Release+Asserts/lib/ -c -g
XXXX.c
```

where XXXX.c is the peripheral driver source file. It is important to note that KLEE executes on a Linux 64-bit platform but the Cortex M3 is a 32-bit processor, so the –m32 option is included to produce 32-bit bitcode [8]. By using 32-bit bitcode, variables of type long and unsigned long are 32-bits instead of 64-bits.

In the previous section I mentioned that KLEE does not support inline assembly. All of the functions in the CPU driver contain inline assembly. These are used to enable/disable interrupts, wait for interrupts to occur, and to get/set the interrupt priority masking level. The SysCtlDelay function in the System Controller driver also contains inline assembly and is used to add a fixed number of delay loop iterations during execution. For these

functions that do not return a value (void), I replaced them with a function that is empty. For the functions that have a return value (unsigned long), I replaced them with a function that just returns 0. These changes were made to prevent KLEE from reporting execution errors (test.exec.err) related to inline assembly. These changes did not induce any side-effects or unexpected behavior on the few functions that called them.

At this point all of the source code under test should be compiled into bitcode object files (.o). In order to test the functions I created an individual .c file for each function under test which would have a main function that calls the function under test with its symbolic arguments. KLEE performs symbolic execution on program variables that are made symbolic. To make a program variable symbolic the program must call the klee_make_symbolic function and pass in the variable's address, size, and name. An example of this call is listed below.

```
1: unsigned long ulBase;
2: klee_make_symbolic(&ulBase, sizeof(ulBase), "ulBase");
```

For each function evaluated, its parameters were treated as symbolic variables. The exception to this case was parameters that were pointers to the address of interrupt handlers. The pointers to the interrupt handlers were treated as concrete variables in these cases.

The Cortex M3 processor communicates with the peripherals through memory-mapped I/O [8]. The registers for each peripheral can be accessed and modified through load/store commands to memory addresses assigned to that peripheral device. These memory addresses needed to be made available to KLEE during execution. In order to allocate specific blocks of memory, the program needs to call the klee_define_fixed_object function with the starting memory location and the number of bytes to allocate. An example of this call is listed below.

```
1: klee_define_fixed_object((void *)(0x40054000), 4096);
```

In the example above a block of 4KB are allocated starting at memory location 0x40054000, which is the base address for the I2S peripheral. The hw_mem_map header file provided the definitions for the base addresses of the memories and peripherals. Some peripherals reference other peripherals in their source code. These peripherals must allocate space for the referenced peripherals as well. For example, the usb driver includes the udma driver in its source, so 4KB are allocated at memory location 0x400FF000, which is the base address for the uDMA peripheral.

For functions that had assertions, I called the klee_assume function with the assertion argument so that the symbolic values generated by KLEE would pass the assertion. In my evaluation I did not generate test cases that would knowingly/intentionally result in assertion violations. This way any assertion violations that were reported by KLEE would be true assertion violations that need addressing.

For functions that have parameters which are pointers to other variables, I created a symbolic array of varying size and passed the name of the array to the function call for that pointer. The size of the array was determined by inspecting the code under test to see how much space needed to be allocated for the data structure.

The only thing left to add to the function under test file was the call to the function. The function under test code for the I2SRxDataGet function is listed below.

```
1: #include "inc/hw_i2s.h"
2: #include "inc/hw_ints.h"
3: #include "inc/hw_memmap.h"
4: #include "inc/hw_types.h"
5: #include "driverlib/debug.h"
6: #include "driverlib/i2s.h"
7: #include "driverlib/interrupt.h"
8: #include <klee/klee.h>
9:
10: void handler(void) {}
11:
12: int main() {
13:   unsigned long ulBase;
14:   klee_make_symbolic(&ulBase, sizeof(ulBase), "ulBase");
15:   unsigned long pulData[1];
16:   klee_make_symbolic(&pulData, sizeof(pulData), "pulData");
```

```
17:
18:    klee_define_fixed_object((void *)(0x40054000), 4096); //I2S
19:    klee_define_fixed_object((void *)(0x00000000), 262144); //FLASH
20:    klee_define_fixed_object((void *)(0x20000000), 65536); //SRAM
21:    klee_define_fixed_object((void *)(0xE0000000), 266240); //NVIC
22:
23:    klee_assume(ulBase == I2S0_BASE);
24:
25:    I2SRxDataGet(ulBase, pulData);
26:    return 0;
27: }
```

I2SRxDataGet .c, function under test file for I2SRxDataGet.

The function under test file is now ready to be compiled into LLVM bitcode for KLEE to test.

```
llvm-gcc -m32 --emit-llvm -I/root/KLEE/klee/include -
I/root/StellarisWare -L /root/KLEE/klee/Release+Asserts/lib/ -c -g
XXXX.c
```

where XXXX is the name of the function under test.

After the object file has been created it needs to be linked with other object files to create the final bitcode used by KLEE during execution.

```
llvm-link /root/StellarisWare/driverlib/YYYY.o
/root/StellarisWare/driverlib/interrupt.o
/root/StellarisWare/driverlib/cpu.o XXXX.o -o XXXX.bc
```

where XXXX is the name of the function under test, and YYYY is the peripheral driver the function under test is defined in. All of the drivers reference functions defined in the interrupt, and cpu drivers, so those object files must be linked in to the bitcode file. The hibernate and usb drivers reference functions in the sysctl driver, so the sysctl object file is linked into the functions under test for those drivers. The usb driver also references functions in the udma driver, so the udma object file is linked in to functions under test for the usb driver as well.

After the bitcode object files have been linked, the resulting bitcode is executed by KLEE with the following command.

```
klee --emit-all-errors --max-time=3600 --watchdog XXXX.bc
```

where XXXX is the name of the function under test. Each function is ran for at most one hour. I used the default search heuristic when running KLEE, which is the random-path selection interleaved with non-uniform random search (nurs) with coverage-new heuristic.

I wrote a Java application that would automate this entire process and execute KLEE for all 554 functions. As the KLEE program executed, it generated the test cases for all execution paths that were evaluated within the one hour time limit. The error messages generated are written to KLEE_Errors.txt and the total test cases generated are written to KLEE_TestCases.txt. The Java application is included in Appendix A. The "includes" being referenced contain the same header files that are in the peripheral driver. The "files" being referenced contain the function definition with all of the assertions following it for that function. Below is a snippet of the adc file for two of the functions.

```
1: ADCIntRegister(unsigned long ulBase, unsigned long ulSequenceNum,
void (*pfnHandler)(void))
2: klee_assume((ulBase == ADC0_BASE) | (ulBase == ADC1_BASE));
3: klee_assume(ulSequenceNum < 4);
4:
5: ADCIntUnregister(unsigned long ulBase, unsigned long ulSequenceNum)
6: klee_assume((ulBase == ADC0_BASE) | (ulBase == ADC1_BASE));
7: klee_assume(ulSequenceNum < 4);
```

# 6. EVALUATION RESULTS

There are twenty-three software modules in the Stellaris Peripheral Driver Library. Twenty-two of the twenty-three software modules were evaluated. All of the functions in the CPU module contained inline assembly, so the CPU module was not evaluated. The total number of test cases generated for each function is listed in Appendix B. The appendix includes all functions within the drivers that were evaluated. There are some functions that have a value set to "void". These functions are functions that have no input parameters (void), and thus were not evaluated.

## 6.1 Function with Two Execution Paths

The source code of the functions varied in complexity and input parameters, so there is a wide range of test cases that were reported for each function. For example, the CAN driver has a function CANBaseValid which validates if the base address argument is a valid CAN controller base address. The code for this function is shown below. As you can see, there are only two paths through this function. One path where the base address is equal to one of the three valid CAN controller base addresses (CAN0_BASE, CAN1_BASE, or CAN2_BASE). The other path is where the base address is not equal to one of the three valid CAN controller base addresses. The test cases that KLEE produced were ulBase equal to 0x40040000 (CAN0_BASE) and ulBase equal to 0x00000000. The first test case would return true and the second would return false.

```
1: tBoolean
2: CANBaseValid(unsigned long ulBase)
3: {
4:     return((ulBase == CAN0_BASE) || (ulBase == CAN1_BASE) ||
5:            (ulBase == CAN2_BASE));
6: }
```

## 6.2 Function with an Assertion

Another simple but interesting example is described in the CANDisable function. This code does not have any complex logic, and appears to only have a single path so you would think that only one test case would be generated. However, there is an assertion that the CAN controller base address is valid. The test cases that KLEE produced were ulBase equal to 0x40040000 (CAN0_BASE), ulBase equal to 0x40041000 (CAN1_BASE), and ulBase equal to 0x40042000 (CAN2_BASE). Most of the drivers in the library have similar functions which validate the base address of the peripheral and other functions that call the validating function to check the inputs to the function.

```
1: void
2: CANDisable(unsigned long ulBase)
3: {
4:     //
5:     // Check the arguments.
6:     //
7:     klee_assert(CANBaseValid(ulBase));
8:
9:     //
10:     // Set the init bit in the control register.
11:     //
12:     CANRegWrite(ulBase + CAN_O_CTL,
13:                 CANRegRead(ulBase + CAN_O_CTL) | CAN_CTL_INIT);
14: }
```

## 6.3 Bugs Found in the Driver Library

The automatic test case generation that KLEE performs is a valuable asset for software engineers. Another amazing feature that KLEE provides is the ability to find bugs. As KLEE performs symbolic execution to generate test cases, it also checks for bugs in the software. Some of which are assertion violations, memory access violations, and

arithmetic errors. The bugs found during analysis of the Stellaris Peripheral Driver Library are listed in Table 2. Each of the errors is described below and a solution is provided to resolve the error for most of the reported errors. The memory errors and divide-by-zero errors were confirmed on the Stellaris EKK-LM3S8962 board by calling the functions with the test case inputs that were generated to produce the error.
For the divide-by-zero errors, in order for the fault to be observed by the Cortex M3 processor, the DIV_0_TRP bit in the Configuration and Control Register (CCR) must be enabled (set to 1) [9]. If it is not set, then the Usage Fault will not be thrown and the Interrupt Default Handler (IntDefaultHandler) will not be called. In this case the quotient is set to 0 and execution resumes as if the operation was valid. The IntDefaultHandler is just an infinite while loop which maintains the system state for debugging purposes. The function call SSIConfigSetExpClk(0x40008000,0xFFFFFFFF,0,0,0,0) and SysCtlI2SMClkSet(0xFFFFFFFF, 0x40000000) will result in the divide-by-zero error when re-ran on the EK-LM3S8962. The optimizations for the compiler need to be disabled in order for the SDIV and UDIV instructions to be produced.

The memory errors reported for the functions SysCtlPeripheralDeepSleepDisable, SysCtlPeripheralDeepSleepEnable, SysCtlPeripheralDisable, SysCtlPeripheralEnable, SysCtlPeripheralReset, SysCtlPeripheralSleepDisable, SysCtlPeripheralSleepEnable are all related. The SysCtlPeripheralDeepSleepDisable function is listed below. This function has an assertion to check that the peripheral identifier is valid. For peripheral identifiers whose peripheral index is not 15 (false condition at line 13), the statement at line 26 is executed. The peripheral identifier (ulPeripheral) is passed to the macro

| Function | Error |
|---|---|
| CANDataRegRead | can.c:264: ASSERTION FAIL: ulIntNumber != (unsigned long)-1 |
| CANDataRegWrite | can.c:402: memory error: out of bound pointer |
| CANMessageSet | KLEE failure |
| CANRegRead | can.c:284: memory error: out of bound pointer |
| CANRegWrite | KLEE failure |
| EthernetPacketGetInternal | KLEE failure |
| EthernetPHYPowerOff | ethernet.c:1305: ASSERTION FAIL: ulBase == ETH_BASE |
| EthernetPHYPowerOn | ethernet.c:1305: ASSERTION FAIL: ulBase == ETH_BASE |
| FlashProtectGet | KLEE failure |
| IntRegister | IntRegister.c:36: invalid klee_assume call (provably false) |
| MPURegionSet | MPURegionSet.c:25: overshift error |
| PWMGenFaultClear | PWMGenFaultClear.c:50: invalid klee_assume call (provably false) |
| PWMGenFaultTriggerSet | PWMGenFaultTriggerSet.c:50: invalid klee_assume call (provably false) |
| SSIConfigSetExpClk | SSIConfigSetExpClk.c:39: divide by zero |
| SysCtlI2SMClkSet | sysctl.c:3106: divide by zero |
| SysCtlPeripheralDeepSleepDisable | sysctl.c:1312: memory error: out of bound pointer |
| SysCtlPeripheralDeepSleepEnable | sysctl.c:1233: memory error: out of bound pointer |
| SysCtlPeripheralDisable | sysctl.c:1005: memory error: out of bound pointer |
| SysCtlPeripheralEnable | sysctl.c:937: memory error: out of bound pointer |
| SysCtlPeripheralReset | sysctl.c:849: memory error: out of bound pointer |
| SysCtlPeripheralSleepDisable | sysctl.c:1156: memory error: out of bound pointer |
| SysCtlPeripheralSleepEnable | sysctl.c:1080: memory error: out of bound pointer |
| UARTConfigGetExpClk | KLEE failure |
| uDMAIntUnregister | interrupt.c:559: ASSERTION FAIL: ulInterrupt < NUM_INTERRUPTS |
| USBEndpointDMADisable | KLEE failure |
| USBEndpointDMAEnable | KLEE failure |
| USBFIFOConfigGet | usb.c:188: ASSERTION FAIL: (ulEndpoint == 0) \|\| (ulEndpoint == 1) \|\| (ulEndpoint == 2) \|\| (ulEndpoint == 3) |
| USBFIFOConfigSet | usb.c:124: ASSERTION FAIL: (ulEndpoint == 0) \|\| (ulEndpoint == 1) \|\| (ulEndpoint == 2) \|\| (ulEndpoint == 3) |
| USBIndexRead | KLEE failure |
| USBIndexWrite | KLEE failure |
| USBPHYPowerOff | KLEE failure |
| USBPHYPowerOn | KLEE failure |

Table 2. KLEE reported errors found during analysis of the Stellaris Peripheral Driver Library.

SYSCTL_PERIPH_INDEX. This macro does a right shift on the least 28 bits of the peripheral identifier to get the 4 most significant bits. This value can range from 0 to 15 (0xF). The value returned by the macro is then used as an index into the g_pulDCGCRegs array, but the array only has a size of 3. If a value of 3 to 15 is returned then a memory violation will occur.

```
1: void
2: SysCtlPeripheralDeepSleepDisable(unsigned long ulPeripheral)
3: {
4:     //
5:     // Check the arguments.
6:     //
7:     klee_assert(SysCtlPeripheralValid(ulPeripheral));
8:
9:     //
10:     // See if the peripheral index is 15, indicating a peripheral
that is
11:     // accessed via the SYSCTL_DCGCperiph registers.
12:     //
13:     if((ulPeripheral & 0xf0000000) == 0xf0000000)
14:     {
15:         //
16:         // Disable this peripheral in deep-sleep mode.
17:         //
18:         HWREGBITW(SYSCTL_DCGCBASE + ((ulPeripheral & 0xff00) >> 8),
19:                   ulPeripheral & 0xff) = 0;
20:     }
21:     else
22:     {
23:         //
24:         // Disable this peripheral in deep-sleep mode.
25:         //
26:         HWREG(g_pulDCGCRegs[SYSCTL_PERIPH_INDEX(ulPeripheral)]) &=
27:             ~SYSCTL_PERIPH_MASK(ulPeripheral);
28:     }
29: }

#define SYSCTL_PERIPH_INDEX(a)  (((a) >> 28) & 0xf)

const unsigned long g_pulDCGCRegs[] = { SYSCTL_DCGC0, SYSCTL_DCGC1,
SYSCTL_DCGC2 };
```

If the valid peripheral identifiers SYSCTL_PERIPH_PLL (0x30000010), SYSCTL_PERIPH_TEMP (0x30000020), or SYSCTL_PERIPH_MPU (0x30000080) are passed as an argument to SysCtlPeripheralDeepSleepDisable, SysCtlPeripheralDeepSleepEnable, SysCtlPeripheralDisable, SysCtlPeripheralEnable, SysCtlPeripheralReset, SysCtlPeripheralSleepDisable, or SysCtlPeripheralSleepEnable, then a memory fault will occur. The processor will then call the fault handler (FaultISR), which sits in an infinite loop.

CANRegRead has an indirect assertion on its input parameter ulRegAddress. The problem with this assertion comes from masking off the 12 least significant bits of ulRegAddress. The test case that was produced that would cause this memory violation has the value of 0x40041FFD for ulRegAddress. This value will pass the assertion, but is not a valid 32-bit aligned address and thus will cause a memory error when HWREG(ulRegAddress) is called. An additional assertion should be added to ensure that ulRegAddress is a multiple of 4: klee_assume(!(ulRegAddress & 3));

```
1: ulIntNumber = CANIntNumberGet(ulRegAddress & 0xfffff000);
2:
3: klee_assert(ulIntNumber != (unsigned long)-1);
```

CANRegWrite suffers from not having an assertion on the ulRegAddress parameter. Since there is no assertion on the register address, KLEE will generate a memory address that will cause the statement HWREG(ulRegAddress) to produce a memory error: out of bound pointer.

For EthernetPHYPowerOff and EthernetPHYPowerOn, there does not exist an assertion on ulBase, but these functions call EthernetPHYWrite which has an assertion on ulBase.

Therefore it is possible for KLEE to generate a test case for EthernetPHYPowerOff/ EthernetPHYPowerOn that causes an assertion violation. To resolve this issue, klee_assert(ulBase == ETH_BASE) should be added to the start of these functions. All of the other functions in the ethernet driver have this assertion, so I am not sure why it was not included for these two functions.

For EthernetPacketGetInternal, there are no assertions on the input parameters. In this case KLEE failed while trying to generate test cases. To resolve this issue, assertions should be added on the input parameters. It's possible that the developers did not include the assertions since this is a static/internal function, but the other get functions, EthernetPacketGet and EthernetPacketGetNonBlocking, have the same assertions on the inputs (klee_assume(ulBase == ETH_BASE); klee_assume(pucBuf != 0); klee_assume(lBufLen > 0);) If these assertions are added, then KLEE is able to create test cases for this function.

PWMGenFaultClear and PWMGenFaultTriggerSet both have the same assertion statements which are contradictory. It is not possible for the ulGroup to be equal to PWM_FAULT_GROUP_0 and PWM_FAULT_GROUP_1 at the same time. This contradiction is made at line 2 and line 5.

```
1: klee_assert((ulGroup == PWM_FAULT_GROUP_0) || (ulGroup ==
PWM_FAULT_GROUP_1));
2: klee_assert((ulGroup == PWM_FAULT_GROUP_0) &&
3:        ((ulFaultTriggers & ~(PWM_FAULT_FAULT0 | PWM_FAULT_FAULT1 |
4:                                              PWM_FAULT_FAULT2 |
PWM_FAULT_FAULT3)) == 0));
5: klee_assert((ulGroup == PWM_FAULT_GROUP_1) &&
6:        ((ulFaultTriggers & ~(PWM_FAULT_DCMP0 | PWM_FAULT_DCMP1 |
7:                                              PWM_FAULT_DCMP2 |
PWM_FAULT_DCMP3 |
```

```
8:                                                  PWM_FAULT_DCMP4 |
PWM_FAULT_DCMP5 |

9:                                                  PWM_FAULT_DCMP6 |

PWM_FAULT_DCMP7)) == 0));
```

For USBFIFOConfigGet and USBFIFOConfigSet, there is an assertion on the ulEndpoint parameter which allows it to have a value in the range USB_EP_1 (x10) to USB_EP_15 (xF0). This value is shifted four bits to the right and passed in to a call to USBIndexRead for the ulEndpoint argument. Once shifted the range of values is 1 to 15. USBIndexRead has an assertion that the value of ulEndpoint be 0 to 3. Therefore an assertion violation can occur if USBFIFOConfigGet or USBFIFOConfigSet is called with ulEndpoint set to USB_EP_4 to USB_EP_15.

For USBPHYPowerOff and USBPHYPowerOn, there is not assertion on the ulBase input parameter. This can result in a memory violation when the memory address at ulBase is read for an address that is not available: HWREGB(ulBase + USB_O_POWER). To resolve this issue, an assertion should be placed on ulBase, klee_assert(ulBase == USB0_BASE). All of the other functions in the library include this assertion, so I am not sure why the developers left this out. These functions are not static/internal functions. Not only is the ulBase assertion missing from USBEndpointDMADisable and USBEndpointDMAEnable, but so is the assertion on ulEndpoint.

For USBIndexRead and USBIndexWrite there is a similar issue, but the assertion needs to be added for the ulIndexedReg parameter. A call is made to HWREGB(ulBase + ulIndexedReg) which could result in a memory access violation.

For uDMAIntUnregister there is no assertion on ulIntChannel. This function calls IntUnregister, which has an assertion on ulInterrupt. To overcome this error, the same assertion called in IntUnregister should be called in uDMAIntUnregister: klee_assume(ulIntChannel < NUM_INTERRUPTS);

## 6.4 Future Research

KLEE is known for generating high-coverage test cases [3]. The code coverage is typically measured by compiling the source code under test with gcov and rerunning the test cases natively (independent of KLEE). The Keil tool chain that I used for compiling/loading the EK-LM3S8962 evaluation board required additional hardware (ULINK Pro) in order to measure code coverage. Future research might include using a different tool chain, perhaps the Mentor Graphics Sourcery CodeBench for Stellaris EABI for a Linux platform.

The bugs I found were passed on to the Texas Instruments StellarisWare developers. I have not heard a response yet. According to TI's website, there will no longer be releases of StellarisWare unless there are major issues [11]. I feel like some of the bugs I found could be major issues, but I will let the developers decide if they want to release the fixes. TI is pushing the use of the Tiva family of microcontrollers and their TivaWare software. A future project could be obtaining a Tiva evaluation board and applying KLEE to TivaWare to generate test cases and find bugs.

# 7. CONCLUSION

Symbolic execution is a programming analysis technique that has been around since the 1970s. Tools based on symbolic execution have been created to automate the testing of software which improves the quality of software. These tools allow software engineers to run exhaustive tests on their software to provide verification and validation. KLEE is an open-source dynamic test generation tool based on symbolic execution that is fairly straightforward to use. KLEE is an ideal test generation tool because it performs symbolic execution on C code that runs on Linux distributions. KLEE can help reduce the time spent checking software by exhaustively generating test cases automatically. KLEE can also help to eliminate maintenance costs by uncovering bugs within our software. The benefits of using KLEE for automated software testing will be observed immediately. For my evaluation, I was able to generate over 14000 test cases on over 550 functions within the Stellaris Peripheral Driver Library. I was able to find 32 bugs in the library as well. KLEE should definitely be put to use by low-level embedded systems engineers to support their testing efforts.

# Appendix A

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;
import java.nio.file.Files;

import org.apache.commons.io.FileUtils;
import org.apache.commons.io.FilenameUtils;
import org.apache.commons.io.IOUtils;

public class makeKLEEinputs {

	public static void main(String[] args) {
		File dir = new File("/root/workspace/makeKLEEinputs/files");
		File outputFile = new
File("/root/workspace/makeKLEEinputs/KLEE_Errors.txt");
		File[] directoryListing = dir.listFiles();
		String pathToDriverLibrary = "/root/StellarisWare/driverlib/";

		try {
			PrintStream out = new PrintStream(new
FileOutputStream("/root/workspace/makeKLEEinputs/KLEE_TestCases.txt"));
			System.setOut(out);
		} catch (FileNotFoundException e1) {
			e1.printStackTrace();
		}

		//Process each driver in the Stellaris Peripheral Driver Library
(except CPU)
		for(File file : directoryListing)
		{
			try {
				FileReader fileReader = new FileReader(file);
				String basename =
FilenameUtils.getBaseName(file.getName()).toLowerCase();
				File directory = new File("functions/" + basename);
				if (!directory.exists()) {
					directory.mkdir();
				}

				BufferedReader bufferedReader = new
BufferedReader(fileReader);
				String line;
				while((line = bufferedReader.readLine()) != null)
				{
```

```
                                boolean hasPtrParameter = false;
                                int index = line.indexOf('(');
                                String functionName = line.substring(0,
index);

                                String path = "functions/" + basename + "/" +
functionName;
                                String pathAndFunctionName = path + "/" +
functionName;

                                directory = new File(path);
                                if (!directory.exists()) {
                                      directory.mkdir();
                                }

                                String[] params =
line.substring(index+1).split(",");

                                File functionFile = new
File(pathAndFunctionName + ".c");
                                Files.copy( new File("includes/" + basename +
".c").toPath(), functionFile.toPath() );

                                StringBuilder mainFunction = new
StringBuilder();
                                mainFunction.append("\r\nint main() {\r\n");

                                for(int i = 0; i < params.length; i++)
                                {
                                      if(i == params.length - 1)
                                      {
                                            params[i] =
params[i].substring(0, params[i].length() - 1);
                                      }
                                      if(params[i].contains("(void)"))
                                      {
                                            params[i] = "handler";
                                      }
                                      else if(params[i].contains("*"))
                                      {
                                            hasPtrParameter = true;
                                      }
                                      else
                                      {
                                            mainFunction.append("\t");
                                            mainFunction.append(params[i]);
                                            mainFunction.append(";\r\n\t");

      mainFunction.append("klee_make_symbolic(&");
                                            String[] paramString =
params[i].split(" ");
```

```
                                        String param =
paramString[paramString.length - 1];
                                        mainFunction.append(param);
                                        mainFunction.append(",
sizeof(");
                                        mainFunction.append(param);
                                        mainFunction.append("), \"");
                                        mainFunction.append(param);
                                        mainFunction.append("\");\r\n");
                                }
                        }

                        if(basename.equals("adc"))
                        {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40038000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40039000), 4096);");
                        }
                        else if(basename.equals("can"))
                        {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40040000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40041000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40042000), 4096);");
                        }

                        else if(basename.equals("comp"))
                        {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4003C000), 4096);");
                        }
                        else if(basename.equals("epi"))
                        {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400D0000), 4096);");
                        }
                        else if(basename.equals("ethernet"))
                        {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40048000), 4096);");
```

```
                                }
                                else if(basename.equals("flash"))
                                {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FD000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FE000), 4096);"); //sysctl (CLASS_IS_SANDSTORM)
                                }
                                else if(basename.equals("gpio"))
                                {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40004000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40005000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40006000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40007000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40024000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40025000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40026000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40027000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4003D000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40058000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40059000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4005A000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4005B000), 4096);");
```

```
      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4005C000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4005D000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4005E000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4005F000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40060000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40061000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40062000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40063000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40064000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40065000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40066000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FE000), 4096);"); //sysctl (CLASS_IS_SANDSTORM)
                              }
                              else if(basename.equals("hibernate"))
                              {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FC000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FE000), 4096);"); //sysctl

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x43FC6000), 65536);"); //sysctl
                              }
                              else if(basename.equals("i2c"))
                              {
```

```
      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40020000), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40020800), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40021000), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40021800), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40022000), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40022800), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40023000), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40023800), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400C0000), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400C0800), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400C1000), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400C1800), 2048);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FE000), 4096);"); //sysctl (CLASS_IS_DUSTDEVIL)
                                }
                                else if(basename.equals("i2s"))
                                {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40054000), 4096);");
                                }
                                else if(basename.equals("pwm"))
                                {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40028000), 4096);");
```

```
      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40029000), 4096);");
                                    }
                                    else if(basename.equals("qei"))
                                    {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4002C000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4002D000), 4096);");
                                    }
                                    else if(basename.equals("ssi"))
                                    {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40008000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40009000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4000A000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4000B000), 4096);");
                                    }
                                    else if(basename.equals("sysctl"))
                                    {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FE000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x43FC6000), 65536);");
                                    }
                                    else if(basename.equals("timer"))
                                    {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40030000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40031000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40032000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40033000), 4096);");
```

```
      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40034000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40035000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40036000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40037000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4004C000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4004D000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4004E000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4004F000), 4096);");

                                }
                                else if(basename.equals("uart"))
                                {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4000C000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4000D000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4000E000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x4000F000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40010000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40011000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40012000), 4096);");
```

```
      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40013000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FE000), 4096);"); //sysctl (CLASS_IS_SANDSTORM)
                                }
                                else if(basename.equals("udma"))
                                {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FF000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FE000), 4096);"); //sysctl (CLASS_IS_SANDSTORM)
                                }
                                else if(basename.equals("usb"))
                                {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40050000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FE000), 4096);");  //sysctl

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x43FC6000), 65536);"); //sysctl

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x400FF000), 4096);");  //udma

                                }
                                else if(basename.equals("watchdog"))
                                {

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40000000), 4096);");

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x40001000), 4096);");
                                }


      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x00000000), 262144);"); //256KB FLASH

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0x20000000), 65536);"); //64KB SRAM

      mainFunction.append("\r\n\tklee_define_fixed_object((void
*)(0xE0000000), 266240);"); //NVIC
```

```
                                mainFunction.append("\r\n\r\n");

                                while(true)
                                {
                                        line = bufferedReader.readLine();
                                        if(line == null || line.equals(""))
                                        {
                                                break;
                                        }
                                        mainFunction.append("\t");
                                        mainFunction.append(line);
                                        mainFunction.append("\r\n");
                                }

                                mainFunction.append("\r\n\t");
                                mainFunction.append(functionName);
                                mainFunction.append("(");

                                for(int i = 0; i < params.length; i++)
                                {
                                        String[] paramString =
params[i].split(" ");
                                        String param =
paramString[paramString.length - 1];
                                        mainFunction.append(param);
                                        if(i != params.length - 1)
                                        {
                                                mainFunction.append(", ");
                                        }
                                }

                                mainFunction.append(");\r\n\treturn
0;\r\n}");

                                if(hasPtrParameter)
                                {
                                        mainFunction = new StringBuilder();
                                        mainFunction.delete(0,
mainFunction.length());

        mainFunction.append(FileUtils.readFileToString(new
File("functionsWithPointerParameters/" + functionName + ".c")));

        FileUtils.writeStringToFile(functionFile, mainFunction.toString(),
false);
                                }
                                else
                                {
```

```
      FileUtils.writeStringToFile(functionFile, mainFunction.toString(),
true);
                                }

                                System.out.print(functionName + "      ");

                                StringBuilder cmd = new StringBuilder();

                                cmd.append("llvm-gcc -m32 --emit-llvm -
I/root/Desktop/KLEE/klee/include -I/root/StellarisWare -L
/root/Desktop/KLEE/klee/Release+Asserts/lib/ -c -g ");
                                cmd.append(functionFile.toPath());
                                cmd.append(" -o ");
                                cmd.append(pathAndFunctionName);
                                cmd.append(".o");

                                ProcessBuilder pb = new
ProcessBuilder("/bin/bash", "-c", cmd.toString());
                                Process process = pb.start();

                                try {
                                      process.waitFor();
                                } catch (InterruptedException e) {
                                      System.out.println("Error executing
llvm-gcc: " + e.getMessage());
                                      continue;
                                }

                                cmd = new StringBuilder();

                                cmd.append("llvm-link ");
                                cmd.append(pathToDriverLibrary);
                                cmd.append(basename);
                                cmd.append(".o ");
                                if(!basename.equals("interrupt"))
                                {
                                      cmd.append(pathToDriverLibrary);
                                      cmd.append("interrupt.o ");
                                }
                                if(basename.equals("hibernate") ||
basename.equals("usb"))
                                {
                                      cmd.append(pathToDriverLibrary);
                                      cmd.append("sysctl.o ");
                                }
                                if(!basename.equals("cpu"))
                                {
                                      cmd.append(pathToDriverLibrary);
                                      cmd.append("cpu.o ");
                                }
```

```
                                if(basename.equals("usb"))
                                {
                                       cmd.append(pathToDriverLibrary);
                                       cmd.append("udma.o ");
                                }
                                cmd.append(pathAndFunctionName);
                                cmd.append(".o -o ");
                                cmd.append(pathAndFunctionName);
                                cmd.append(".bc");

                                pb = new ProcessBuilder("/bin/bash", "-c",
cmd.toString());
                                process = pb.start();

                                try {
                                       process.waitFor();
                                } catch (InterruptedException e) {
                                       System.out.println("Error executing
llvm-ld: " + e.getMessage());
                                       continue;
                                }

                                //Generate test cases for the function under
test for at most one hour
                                cmd = new StringBuilder();

                                cmd.append("klee --emit-all-errors --max-
time=3600 --watchdog ");
                                cmd.append(pathAndFunctionName);
                                cmd.append(".bc");

                                pb = new ProcessBuilder("/bin/bash", "-c",
cmd.toString());
                                process = pb.start();

                                try
                                {

                            process.waitFor();
                            String message = FileUtils.readFileToString(new
File(path + "/klee-last/messages.txt"));
                            if(!(message.equals("")))
                            {
                                FileUtils.writeStringToFile(outputFile, path
+ "\t" + message, true);
                            }
                                } catch (InterruptedException e) {
                                       System.out.println("Error executing
klee: " + e.getMessage());
                                       continue;
                                } catch (FileNotFoundException e) {
```

```
                                    FileUtils.writeStringToFile(outputFile,
e.getMessage() + "\n", true);
                                }

                                //write total number of test cases generated
to output file
                                cmd = new StringBuilder();

                                cmd.append("ls " + path + "/klee-last/*.ktest
| wc -l");
                                pb = new ProcessBuilder("/bin/bash", "-c",
cmd.toString());
                                pb.redirectErrorStream(true);
                                process = pb.start();
                                IOUtils.copy(process.getInputStream(),
System.out);

                                try
                                {

                            process.waitFor();
                                } catch (InterruptedException e) {
                                    System.out.println("Error executing
klee: " + e.getMessage());
                                    continue;
                                }
                            }
                        }
                        catch (IOException e)
                        {
                            e.printStackTrace();
                        }
                }
                System.out.println("Processing Complete!");
        }


}
```

# Appendix B

| Functions | Number of Test Cases Generated |
|---|---|
| ADCComparatorConfigure | 2 |
| ADCComparatorIntClear | 2 |
| ADCComparatorIntDisable | 2 |
| ADCComparatorIntEnable | 2 |
| ADCComparatorIntStatus | 2 |
| ADCComparatorRegionSet | 2 |
| ADCComparatorReset | 8 |
| ADCHardwareOversampleConfigure | 14 |
| ADCIntClear | 2 |
| ADCIntDisable | 2 |
| ADCIntEnable | 2 |
| ADCIntRegister | 2 |
| ADCIntStatus | 4 |
| ADCIntUnregister | 2 |
| ADCPhaseDelayGet | 2 |
| ADCPhaseDelaySet | 2 |
| ADCProcessorTrigger | 2 |
| ADCReferenceGet | 2 |
| ADCReferenceSet | 2 |
| ADCResolutionGet | 2 |
| ADCResolutionSet | 2 |
| ADCSequenceConfigure | 2 |
| ADCSequenceDataGet | 2 |
| ADCSequenceDisable | 2 |
| ADCSequenceEnable | 2 |
| ADCSequenceOverflow | 2 |
| ADCSequenceOverflowClear | 2 |
| ADCSequenceStepConfigure | 16 |
| ADCSequenceUnderflow | 2 |
| ADCSequenceUnderflowClear | 2 |
| ADCSoftwareOversampleConfigure | 5 |
| ADCSoftwareOversampleDataGet | 22 |
| ADCSoftwareOversampleStepConfigure | 4 |
| CANBaseValid | 2 |

| | |
|---|---|
| CANBitRateSet | 128 |
| CANBitTimingGet | 3 |
| CANBitTimingSet | 3 |
| CANDataRegRead | 2 |
| CANDataRegWrite | 3 |
| CANDisable | 3 |
| CANEnable | 3 |
| CANErrCntrGet | 3 |
| CANInit | 3 |
| CANIntClear | 6 |
| CANIntDisable | 3 |
| CANIntEnable | 3 |
| CANIntNumberGet | 4 |
| CANIntRegister | 3 |
| CANIntStatus | 7 |
| CANIntUnregister | 3 |
| CANMessageClear | 3 |
| CANMessageGet | 6 |
| CANMessageSet | 0 |
| CANRegRead | 6 |
| CANRegWrite | 0 |
| CANRetryGet | 3 |
| CANRetrySet | 6 |
| CANStatusGet | 13 |
| ComparatorConfigure | 1 |
| ComparatorIntClear | 1 |
| ComparatorIntDisable | 1 |
| ComparatorIntEnable | 1 |
| ComparatorIntRegister | 1 |
| ComparatorIntStatus | 2 |
| ComparatorIntUnregister | 1 |
| ComparatorRefSet | 1 |
| ComparatorValueGet | 1 |
| EPIAddressMapSet | 1 |
| EPIConfigGPModeSet | 2 |
| EPIConfigHB16Set | 4 |
| EPIConfigHB8Set | 4 |
| EPIConfigSDRAMSet | 1 |

| | |
|---|---|
| EPIDividerSet | 1 |
| EPIFIFOConfig | 1 |
| EPIIntDisable | 1 |
| EPIIntEnable | 1 |
| EPIIntErrorClear | 1 |
| EPIIntErrorStatus | 1 |
| EPIIntRegister | 1 |
| EPIIntStatus | 2 |
| EPIIntUnregister | 1 |
| EPIModeSet | 1 |
| EPINonBlockingReadAvail | 1 |
| EPINonBlockingReadConfigure | 1 |
| EPINonBlockingReadCount | 1 |
| EPINonBlockingReadGet16 | 1 |
| EPINonBlockingReadGet32 | 1 |
| EPINonBlockingReadGet8 | 1 |
| EPINonBlockingReadStart | 1 |
| EPINonBlockingReadStop | 1 |
| EPIWriteFIFOCountGet | 1 |
| EthernetConfigGet | 1 |
| EthernetConfigSet | 1 |
| EthernetDisable | 1 |
| EthernetEnable | 1 |
| EthernetInitExpClk | 1 |
| EthernetIntClear | 1 |
| EthernetIntDisable | 1 |
| EthernetIntEnable | 1 |
| EthernetIntRegister | 1 |
| EthernetIntStatus | 2 |
| EthernetIntUnregister | 1 |
| EthernetMACAddrGet | 1 |
| EthernetMACAddrSet | 1 |
| EthernetPacketAvail | 1 |
| EthernetPacketGet | 1 |
| EthernetPacketGetInternal | 0 |
| EthernetPacketGetNonBlocking | 1 |
| EthernetPacketPut | 2047 |
| EthernetPacketPutInternal | 1 |

| | |
|---|---|
| EthernetPacketPutNonBlocking | 2047 |
| EthernetPHYAddrSet | 1 |
| EthernetPHYPowerOff | 2 |
| EthernetPHYPowerOn | 2 |
| EthernetPHYRead | 1 |
| EthernetPHYWrite | 1 |
| EthernetSpaceAvail | 1 |
| FlashErase | 1 |
| FlashIntClear | 1 |
| FlashIntDisable | 1 |
| FlashIntEnable | 1 |
| FlashIntRegister | 1 |
| FlashIntStatus | 2 |
| FlashIntUnregister | void |
| FlashProgram | 2 |
| FlashProtectGet | 0 |
| FlashProtectSave | void |
| FlashProtectSet | 3 |
| FlashUsecGet | void |
| FlashUsecSet | 1 |
| FlashUserGet | 1 |
| FlashUserSave | void |
| FlashUserSet | 1 |
| GPIOADCTriggerDisable | 24 |
| GPIOADCTriggerEnable | 24 |
| GPIOBaseValid | 2 |
| GPIODirModeGet | 24 |
| GPIODirModeSet | 72 |
| GPIODMATriggerDisable | 24 |
| GPIODMATriggerEnable | 24 |
| GPIOGetIntNumber | 16 |
| GPIOIntTypeGet | 24 |
| GPIOIntTypeSet | 120 |
| GPIOPadConfigGet | 24 |
| GPIOPadConfigSet | 480 |
| GPIOPinConfigure | 15 |
| GPIOPinIntClear | 24 |
| GPIOPinIntDisable | 24 |

| | |
|---|---|
| GPIOPinIntEnable | 24 |
| GPIOPinIntStatus | 48 |
| GPIOPinRead | 24 |
| GPIOPinTypeADC | 24 |
| GPIOPinTypeCAN | 24 |
| GPIOPinTypeComparator | 24 |
| GPIOPinTypeEPI | 24 |
| GPIOPinTypeEthernetLED | 24 |
| GPIOPinTypeEthernetMII | 24 |
| GPIOPinTypeFan | 24 |
| GPIOPinTypeGPIOInput | 24 |
| GPIOPinTypeGPIOOutput | 24 |
| GPIOPinTypeGPIOOutputOD | 24 |
| GPIOPinTypeI2C | 24 |
| GPIOPinTypeI2CSCL | 24 |
| GPIOPinTypeI2S | 24 |
| GPIOPinTypeLPC | 24 |
| GPIOPinTypePECIRx | 24 |
| GPIOPinTypePECITx | 24 |
| GPIOPinTypePWM | 24 |
| GPIOPinTypeQEI | 24 |
| GPIOPinTypeSSI | 24 |
| GPIOPinTypeTimer | 24 |
| GPIOPinTypeUART | 24 |
| GPIOPinTypeUSBAnalog | 24 |
| GPIOPinTypeUSBDigital | 24 |
| GPIOPinWrite | 24 |
| GPIOPortIntRegister | 15 |
| GPIOPortIntUnregister | 15 |
| HibernateBatCheckDone | void |
| HibernateBatCheckStart | void |
| HibernateClockConfig | 1 |
| HibernateClockSelect | 1 |
| HibernateDataGet | 65 |
| HibernateDataSet | 65 |
| HibernateDisable | void |
| HibernateEnableExpClk | 1 |
| HibernateGPIORetentionDisable | void |

| | |
|---|---|
| HibernateGPIORetentionEnable | void |
| HibernateGPIORetentionGet | void |
| HibernateIntClear | 1 |
| HibernateIntDisable | 1 |
| HibernateIntEnable | 1 |
| HibernateIntRegister | 1 |
| HibernateIntStatus | 2 |
| HibernateIntUnregister | void |
| HibernateIsActive | void |
| HibernateLowBatGet | void |
| HibernateLowBatSet | 1 |
| HibernateRequest | void |
| HibernateRTCDisable | void |
| HibernateRTCEnable | void |
| HibernateRTCGet | void |
| HibernateRTCMatch0Get | void |
| HibernateRTCMatch0Set | 1 |
| HibernateRTCMatch1Get | void |
| HibernateRTCMatch1Set | 1 |
| HibernateRTCSet | 1 |
| HibernateRTCSSGet | void |
| HibernateRTCSSMatch0Get | void |
| HibernateRTCSSMatch0Set | 1 |
| HibernateRTCTrimGet | void |
| HibernateRTCTrimSet | 1 |
| HibernateWakeGet | void |
| HibernateWakeSet | 1 |
| I2CIntNumberGet | 7 |
| I2CIntRegister | 6 |
| I2CIntUnregister | 6 |
| I2CMasterBaseValid | 2 |
| I2CMasterBusBusy | 6 |
| I2CMasterBusy | 6 |
| I2CMasterControl | 6 |
| I2CMasterDataGet | 6 |
| I2CMasterDataPut | 6 |
| I2CMasterDisable | 6 |
| I2CMasterEnable | 6 |

| | |
|---|---|
| I2CMasterErr | 6 |
| I2CMasterInitExpClk | 12 |
| I2CMasterIntClear | 6 |
| I2CMasterIntClearEx | 6 |
| I2CMasterIntDisable | 6 |
| I2CMasterIntDisableEx | 6 |
| I2CMasterIntEnable | 6 |
| I2CMasterIntEnableEx | 6 |
| I2CMasterIntStatus | 12 |
| I2CMasterIntStatusEx | 12 |
| I2CMasterLineStateGet | 6 |
| I2CMasterSlaveAddrSet | 6 |
| I2CMasterTimeoutSet | 6 |
| I2CSlaveACKOverride | 12 |
| I2CSlaveACKValueSet | 12 |
| I2CSlaveAddressSet | 12 |
| I2CSlaveBaseValid | 2 |
| I2CSlaveDataGet | 6 |
| I2CSlaveDataPut | 6 |
| I2CSlaveDisable | 6 |
| I2CSlaveEnable | 6 |
| I2CSlaveInit | 6 |
| I2CSlaveIntClear | 6 |
| I2CSlaveIntClearEx | 6 |
| I2CSlaveIntDisable | 6 |
| I2CSlaveIntDisableEx | 6 |
| I2CSlaveIntEnable | 6 |
| I2CSlaveIntEnableEx | 6 |
| I2CSlaveIntStatus | 12 |
| I2CSlaveIntStatusEx | 12 |
| I2CSlaveStatus | 6 |
| I2SIntClear | 1 |
| I2SIntDisable | 1 |
| I2SIntEnable | 1 |
| I2SIntRegister | 1 |
| I2SIntStatus | 2 |
| I2SIntUnregister | 1 |
| I2SMasterClockSelect | 1 |

| | |
|---|---|
| I2SRxConfigSet | 3 |
| I2SRxDataGet | 1 |
| I2SRxDataGetNonBlocking | 1 |
| I2SRxDisable | 1 |
| I2SRxEnable | 1 |
| I2SRxFIFOLevelGet | 1 |
| I2SRxFIFOLimitGet | 1 |
| I2SRxFIFOLimitSet | 1 |
| I2STxConfigSet | 2 |
| I2STxDataPut | 1 |
| I2STxDataPutNonBlocking | 1 |
| I2STxDisable | 1 |
| I2STxEnable | 1 |
| I2STxFIFOLevelGet | 1 |
| I2STxFIFOLimitGet | 1 |
| I2STxFIFOLimitSet | 1 |
| I2STxRxConfigSet | 3 |
| I2STxRxDisable | 1 |
| I2STxRxEnable | 1 |
| IntDefaultHandler | void |
| IntDisable | 6 |
| IntEnable | 6 |
| IntIsEnabled | 6 |
| IntMasterDisable | void |
| IntMasterEnable | void |
| IntPendClear | 4 |
| IntPendSet | 5 |
| IntPriorityGet | 1 |
| IntPriorityGroupingGet | void |
| IntPriorityGroupingSet | 1 |
| IntPriorityMaskGet | void |
| IntPriorityMaskSet | 1 |
| IntPrioritySet | 1 |
| IntRegister | 1 |
| IntUnregister | 1 |
| MPUDisable | void |
| MPUEnable | 1 |
| MPUIntRegister | 1 |

| | |
|---|---|
| MPUIntUnregister | void |
| MPURegionCountGet | void |
| MPURegionDisable | 1 |
| MPURegionEnable | 1 |
| MPURegionGet | 1 |
| MPURegionSet | 2 |
| PWMDeadBandDisable | 2 |
| PWMDeadBandEnable | 2 |
| PWMFaultIntClear | 2 |
| PWMFaultIntClearExt | 2 |
| PWMFaultIntGet | 2 |
| PWMFaultIntRegister | 2 |
| PWMFaultIntUnregister | 2 |
| PWMGenConfigure | 4 |
| PWMGenDisable | 2 |
| PWMGenEnable | 2 |
| PWMGenFaultClear | 1 |
| PWMGenFaultConfigure | 2 |
| PWMGenFaultStatus | 4 |
| PWMGenFaultTriggerGet | 4 |
| PWMGenFaultTriggerSet | 1 |
| PWMGenIntClear | 2 |
| PWMGenIntGet | 9 |
| PWMGenIntRegister | 8 |
| PWMGenIntStatus | 4 |
| PWMGenIntTrigDisable | 2 |
| PWMGenIntTrigEnable | 2 |
| PWMGenIntUnregister | 8 |
| PWMGenPeriodGet | 2 |
| PWMGenPeriodSet | 2 |
| PWMGenValid | 2 |
| PWMIntDisable | 2 |
| PWMIntEnable | 2 |
| PWMIntStatus | 4 |
| PWMOutputFault | 4 |
| PWMOutputFaultLevel | 4 |
| PWMOutputInvert | 4 |
| PWMOutputState | 4 |

| | |
|---|---|
| PWMOutValid | 2 |
| PWMPulseWidthGet | 4 |
| PWMPulseWidthSet | 4 |
| PWMSyncTimeBase | 2 |
| PWMSyncUpdate | 2 |
| QEIConfigure | 2 |
| QEIDirectionGet | 2 |
| QEIDisable | 2 |
| QEIEnable | 2 |
| QEIErrorGet | 2 |
| QEIIntClear | 2 |
| QEIIntDisable | 2 |
| QEIIntEnable | 2 |
| QEIIntRegister | 2 |
| QEIIntStatus | 4 |
| QEIIntUnregister | 2 |
| QEIPositionGet | 2 |
| QEIPositionSet | 2 |
| QEIVelocityConfigure | 2 |
| QEIVelocityDisable | 2 |
| QEIVelocityEnable | 2 |
| QEIVelocityGet | 2 |
| SSIBaseValid | 2 |
| SSIBusy | 4 |
| SSIClockSourceGet | 4 |
| SSIClockSourceSet | 4 |
| SSIConfigSetExpClk | 265 |
| SSIDataGet | 4 |
| SSIDataGetNonBlocking | 4 |
| SSIDataPut | 4 |
| SSIDataPutNonBlocking | 4 |
| SSIDisable | 4 |
| SSIDMADisable | 4 |
| SSIDMAEnable | 4 |
| SSIEnable | 4 |
| SSIIntClear | 4 |
| SSIIntDisable | 4 |
| SSIIntEnable | 4 |

| | |
|---|---|
| SSIIntNumberGet | 5 |
| SSIIntRegister | 4 |
| SSIIntStatus | 8 |
| SSIIntUnregister | 4 |
| SysCtlADCSpeedGet | void |
| SysCtlADCSpeedSet | 1 |
| SysCtlBrownOutConfigSet | 1 |
| SysCtlClkVerificationClear | void |
| SysCtlClockGet | void |
| SysCtlClockSet | 5 |
| SysCtlDeepSleep | void |
| SysCtlDeepSleepClockSet | 1 |
| SysCtlDelay | void |
| SysCtlFlashSizeGet | void |
| SysCtlGPIOAHBDisable | 1 |
| SysCtlGPIOAHBEnable | 1 |
| SysCtlI2SMClkSet | 4 |
| SysCtlIntClear | 1 |
| SysCtlIntDisable | 1 |
| SysCtlIntEnable | 1 |
| SysCtlIntRegister | 1 |
| SysCtlIntStatus | 2 |
| SysCtlIntUnregister | void |
| SysCtlIOSCVerificationSet | 2 |
| SysCtlLDOConfigSet | 1 |
| SysCtlLDOGet | void |
| SysCtlLDOSet | 1 |
| SysCtlMOSCConfigSet | 1 |
| SysCtlMOSCVerificationSet | 2 |
| SysCtlPeripheralClockGating | 2 |
| SysCtlPeripheralDeepSleepDisable | 3 |
| SysCtlPeripheralDeepSleepEnable | 3 |
| SysCtlPeripheralDisable | 3 |
| SysCtlPeripheralEnable | 3 |
| SysCtlPeripheralMapToNew | 37 |
| SysCtlPeripheralPowerOff | 37 |
| SysCtlPeripheralPowerOn | 37 |
| SysCtlPeripheralPresent | 3 |

| | |
|---|---|
| SysCtlPeripheralReady | 37 |
| SysCtlPeripheralReset | 3 |
| SysCtlPeripheralSleepDisable | 3 |
| SysCtlPeripheralSleepEnable | 3 |
| SysCtlPeripheralValid | 2 |
| SysCtlPinPresent | 1 |
| SysCtlPIOSCCalibrate | 4 |
| SysCtlPLLVerificationSet | 2 |
| SysCtlPWMClockGet | void |
| SysCtlPWMClockSet | 1 |
| SysCtlReset | void |
| SysCtlResetCauseClear | 1 |
| SysCtlResetCauseGet | void |
| SysCtlSleep | void |
| SysCtlSRAMSizeGet | void |
| SysCtlUSBPLLDisable | void |
| SysCtlUSBPLLEnable | void |
| SysTickDisable | void |
| SysTickEnable | void |
| SysTickIntDisable | void |
| SysTickIntEnable | void |
| SysTickIntRegister | 1 |
| SysTickIntUnregister | void |
| SysTickPeriodGet | void |
| SysTickPeriodSet | 1 |
| SysTickValueGet | void |
| TimerBaseValid | 2 |
| TimerConfigure | 768 |
| TimerControlEvent | 12 |
| TimerControlLevel | 24 |
| TimerControlStall | 24 |
| TimerControlTrigger | 24 |
| TimerControlWaitOnTrigger | 72 |
| TimerDisable | 12 |
| TimerEnable | 12 |
| TimerIntClear | 12 |
| TimerIntDisable | 12 |
| TimerIntEnable | 12 |

| | |
|---|---|
| TimerIntNumberGet | 13 |
| TimerIntRegister | 36 |
| TimerIntStatus | 24 |
| TimerIntUnregister | 36 |
| TimerLoadGet | 24 |
| TimerLoadGet64 | 12 |
| TimerLoadSet | 36 |
| TimerLoadSet64 | 12 |
| TimerMatchGet | 24 |
| TimerMatchGet64 | 12 |
| TimerMatchSet | 36 |
| TimerMatchSet64 | 12 |
| TimerPrescaleGet | 24 |
| TimerPrescaleMatchGet | 24 |
| TimerPrescaleMatchSet | 36 |
| TimerPrescaleSet | 36 |
| TimerQuiesce | 12 |
| TimerRTCDisable | 12 |
| TimerRTCEnable | 12 |
| TimerSynchronize | 1 |
| TimerValueGet | 24 |
| TimerValueGet64 | 12 |
| UART9BitAddrSend | 8 |
| UART9BitAddrSet | 8 |
| UART9BitDisable | 8 |
| UART9BitEnable | 8 |
| UARTBaseValid | 2 |
| UARTBreakCtl | 16 |
| UARTBusy | 8 |
| UARTCharGet | 8 |
| UARTCharGetNonBlocking | 8 |
| UARTCharPut | 8 |
| UARTCharPutNonBlocking | 8 |
| UARTCharsAvail | 8 |
| UARTClockSourceGet | 8 |
| UARTClockSourceSet | 8 |
| UARTConfigGetExpClk | 0 |
| UARTConfigSetExpClk | 8 |

| | |
|---|---|
| UARTDisable | 8 |
| UARTDisableSIR | 8 |
| UARTDMADisable | 8 |
| UARTDMAEnable | 8 |
| UARTEnable | 8 |
| UARTEnableSIR | 16 |
| UARTFIFODisable | 8 |
| UARTFIFOEnable | 8 |
| UARTFIFOLevelGet | 8 |
| UARTFIFOLevelSet | 8 |
| UARTFlowControlGet | 8 |
| UARTFlowControlSet | 8 |
| UARTIntClear | 8 |
| UARTIntDisable | 8 |
| UARTIntEnable | 8 |
| UARTIntNumberGet | 9 |
| UARTIntRegister | 8 |
| UARTIntStatus | 16 |
| UARTIntUnregister | 8 |
| UARTModemControlClear | 1 |
| UARTModemControlGet | 1 |
| UARTModemControlSet | 1 |
| UARTModemStatusGet | 1 |
| UARTParityModeGet | 8 |
| UARTParityModeSet | 8 |
| UARTRxErrorClear | 8 |
| UARTRxErrorGet | 8 |
| UARTSmartCardDisable | 8 |
| UARTSmartCardEnable | 8 |
| UARTSpaceAvail | 8 |
| UARTTxIntModeGet | 8 |
| UARTTxIntModeSet | 8 |
| uDMAChannelAssign | 1 |
| uDMAChannelAttributeDisable | 16 |
| uDMAChannelAttributeEnable | 16 |
| uDMAChannelAttributeGet | 1 |
| uDMAChannelControlSet | 1 |
| uDMAChannelDisable | 1 |

| | |
|---|---|
| uDMAChannelEnable | 1 |
| uDMAChannelIsEnabled | 1 |
| uDMAChannelModeGet | 3 |
| uDMAChannelRequest | 1 |
| uDMAChannelScatterGatherSet | 2 |
| uDMAChannelSelectDefault | 1 |
| uDMAChannelSelectSecondary | 1 |
| uDMAChannelSizeGet | 2 |
| uDMAChannelTransferSet | 14 |
| uDMAControlAlternateBaseGet | void |
| uDMAControlBaseGet | void |
| uDMAControlBaseSet | 1 |
| uDMADisable | void |
| uDMAEnable | void |
| uDMAErrorStatusClear | void |
| uDMAErrorStatusGet | void |
| uDMAIntClear | 1 |
| uDMAIntRegister | 1 |
| uDMAIntStatus | void |
| uDMAIntUnregister | 7 |
| USBDevAddrGet | 1 |
| USBDevAddrSet | 1 |
| USBDevConnect | 1 |
| USBDevDisconnect | 1 |
| USBDevEndpointConfigGet | 2 |
| USBDevEndpointConfigSet | 24 |
| USBDevEndpointDataAck | 3 |
| USBDevEndpointStall | 3 |
| USBDevEndpointStallClear | 3 |
| USBDevEndpointStatusClear | 9 |
| USBDevMode | 1 |
| USBEndpointDataAvail | 2 |
| USBEndpointDataGet | 2 |
| USBEndpointDataPut | 5169 |
| USBEndpointDataSend | 2 |
| USBEndpointDataToggleClear | 2 |
| USBEndpointDMAChannel | 1 |
| USBEndpointDMADisable | 0 |

| | |
|---|---|
| USBEndpointDMAEnable | 0 |
| USBEndpointStatus | 1 |
| USBFIFOAddrGet | 1 |
| USBFIFOConfigGet | 4 |
| USBFIFOConfigSet | 4 |
| USBFIFOFlush | 3 |
| USBFrameNumberGet | 1 |
| USBHostAddrGet | 2 |
| USBHostAddrSet | 2 |
| USBHostEndpointConfig | 98 |
| USBHostEndpointDataAck | 2 |
| USBHostEndpointDataToggle | 6 |
| USBHostEndpointStatusClear | 2 |
| USBHostHubAddrGet | 2 |
| USBHostHubAddrSet | 6 |
| USBHostMode | 1 |
| USBHostPwrConfig | 1 |
| USBHostPwrDisable | 1 |
| USBHostPwrEnable | 1 |
| USBHostPwrFaultDisable | 1 |
| USBHostPwrFaultEnable | 1 |
| USBHostRequestIN | 2 |
| USBHostRequestINClear | 2 |
| USBHostRequestStatus | 1 |
| USBHostReset | 2 |
| USBHostResume | 2 |
| USBHostSpeedGet | 1 |
| USBHostSuspend | 1 |
| USBIndexRead | 0 |
| USBIndexWrite | 0 |
| USBIntDisable | 32 |
| USBIntDisableControl | 8 |
| USBIntDisableEndpoint | 1 |
| USBIntEnable | 32 |
| USBIntEnableControl | 8 |
| USBIntEnableEndpoint | 1 |
| USBIntRegister | 1 |
| USBIntStatus | 1 |

| | |
|---|---|
| USBIntStatusControl | 1 |
| USBIntStatusEndpoint | 1 |
| USBIntUnregister | 1 |
| USBModeGet | 1 |
| USBNumEndpointsGet | 1 |
| USBOTGMode | 1 |
| USBOTGSessionRequest | 2 |
| USBPHYPowerOff | 0 |
| USBPHYPowerOn | 0 |
| WatchdogEnable | 2 |
| WatchdogIntClear | 2 |
| WatchdogIntEnable | 2 |
| WatchdogIntRegister | 1 |
| WatchdogIntStatus | 4 |
| WatchdogIntTypeSet | 2 |
| WatchdogIntUnregister | 1 |
| WatchdogLock | 2 |
| WatchdogLockState | 2 |
| WatchdogReloadGet | 2 |
| WatchdogReloadSet | 2 |
| WatchdogResetDisable | 2 |
| WatchdogResetEnable | 2 |
| WatchdogRunning | 2 |
| WatchdogStallDisable | 2 |
| WatchdogStallEnable | 2 |
| WatchdogUnlock | 2 |
| WatchdogValueGet | 2 |

# References


1. J. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.
2. C. Cadar, P. Godefroid, S. Khurshid, C. Pasareanu, K. Sen, N. Tillmann, W. Visser. Symbolic Execution for Software Testing in Practice – Preliminary Assessment. In *ICSE'2011*, Honolulu, HI, USA, May 2011.
3. C. Cadar, D. Dunbar, and D. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2008.
4. The KLEE Symbolic Virtual Machine, http://klee.llvm.org.
5. Stellaris® Peripheral Driver Library USER'S GUIDE, SW-DRL-UG-10636.pdf
6. StellarisWare® Driver Library Standalone Package, http://www.ti.com/tool/sw-drl
7. S. Khurshid, C. Pasareanu, and W. Visser. Generalized Symbolic Execution for Model Checking and Testing. In *Proc. 9th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Warsaw, Poland, April 2003.
8. Cortex-M3 Processor, http://www.arm.com/products/processors/cortex-m/cortex-m3.php
9. Cortex-M3 Devices Generic User Guide: 4.3.10. Configurable Fault Status Register, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/Cihcfefj.html
10. LM3S8962 Ethernet+CAN Evaluation Kits, http://www.ti.com/tool/ek-lm3s8962
11. StellarisWare® Complete (all boards, all components), http://www.ti.com/tool/sw-lm3s