CIS655/CSE661: Advanced Computer Architecture

Fall 2015

Final Report

Topic: TrustZone

Date: 12/10/2015

Name: Jiayu Li	SUID:426453012
Name: Zhihao Zhang	SUID:604903641
Name: Nuo Xu	SUID:946768867

Content

Abstract: 3
1 Introduction: 3
    1.1 ARM® TrustZone® technology 3
    1.2 Trust-Zone Hardware Architecture 4
2. Background 4
    2.1 Software Architecture 4
    2.2 SMC 6
    2.3 Mechanism 9
3. TrustZone study in DS-5 10
    3.1 Initialization 11
    3.2 Initialize Secure Monitor 13
    3.3 Initialize Non-secure world 14
    3.4Main program 15
    3.5 Conclusion 17
4. Simulating the Mechanism of Trustzone (C++) 19
    4.1 Mechanism of C++ Program Simulating Trustzone 19
    4.2 Development Platform 19
    4.3Development 19
    4.4 Process of Executing Program 21
    4.5 Workflow Chart 22
    4.6 Core Code 22
    4.7 Test 25
    4.8 Conclusion 31
5.One-Time Password Use Case 31
    5.1 Context 32
    5.2 Design 32
    5.3 Demo Structure 33
    5.4 Deployment of the Demo in C# 34
    5.5 Activity: 36
    5.6 Results 37
    5.7 Conclusion 41
Reference: 41

Abstract:

ARM® TrustZone® technology is a system-wide approach to security that can be used to protect services and devices from scalable attacks. Today, the new technology is being used in lots of kinds of fields to help people in their life, especially protecting people's properties. So, we are very interested in knowing how the technology works and plan to design an emulator to simulate the mechanism of the TrustZone. However, without the support of related hardware, we firstly know the mechanism of TrustZone by reading the background of TrustZone and running a related program. And then we use the mechanism to develop programs to simulate and implement the mechanism of TrustZone. In this paper, we introduce the background of the TrustZone in the first chapter. In the second chapter, we use DS-5 to build a program about TrustZone so that we can explore the mechanism further. After knowing the mechanism of the TrustZone, in the third chapter, we use the C++ language to build a program to simulate the working principle of TrustZone. Finally, based on what we know about the principle of TrustZone and the experience of building example of TrustZone, we use the C# language to take advantage of TrustZone features to create a security system in the last chapter.

Keywords: TrustZone, emulator, DS-5, C++, C#, Mechanism

1 Introduction:

1.1 ARM® TrustZone® technology

ARM® TrustZone® technology is a system-wide approach to security for a wide array of client and server computing platforms, including handsets, tablets, wearable devices and enterprise systems. It acts as a buffer between the kernel and the hardware. It is incorporated into recent ARM processors, such as ARM11, CortexA8, CortexA9 and CortexA15. To improve security, these ARM processors can run a secure operating system (secure OS) and a normal operating system (normal OS) at the same time from a single core. The instruction Secure Monitor Call or SMC bridges the secure and normal modes.

1.2 Trust-Zone Hardware Architecture

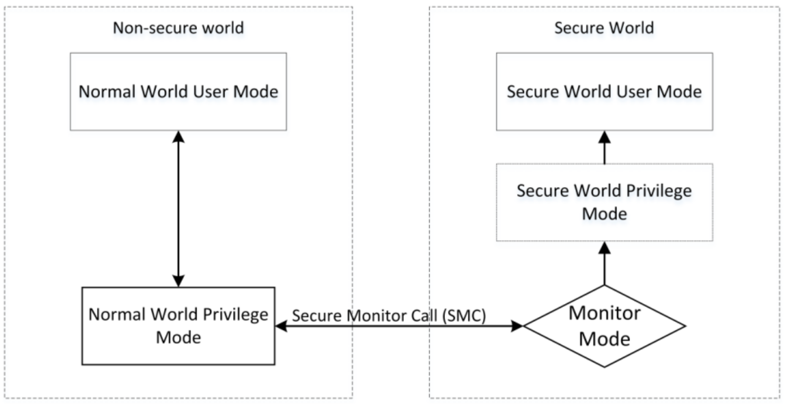


Figure 1.1

As figure 1.1 shows, TrustZone enables a single physical processor core to execute code safely and efficiently from both the Normal world and the Secure world. Software Secure Monitor code, running in the Secure Monitor Mode, links the two worlds and acts as a gatekeeper to manage program flow. The system can have both Secure and Non-secure peripherals that suitable Secure and Non-secure device drivers control.

2. Background

2.1 Software Architecture

Implement security zones in the SoC hardware requires to run some security software in it and make use of the sensitive assets stored in it. There may be many software architectures (Figure 2.1) which based on the processors that support TrustZone and security area on the software stack can be realized. The highest level of software architecture is a specialized security areas of operating system. The simplest way is that a simple library of code in the secure world which can handle one task at a time is sufficient for many applications. There are many Intermediate options between these two extremes.

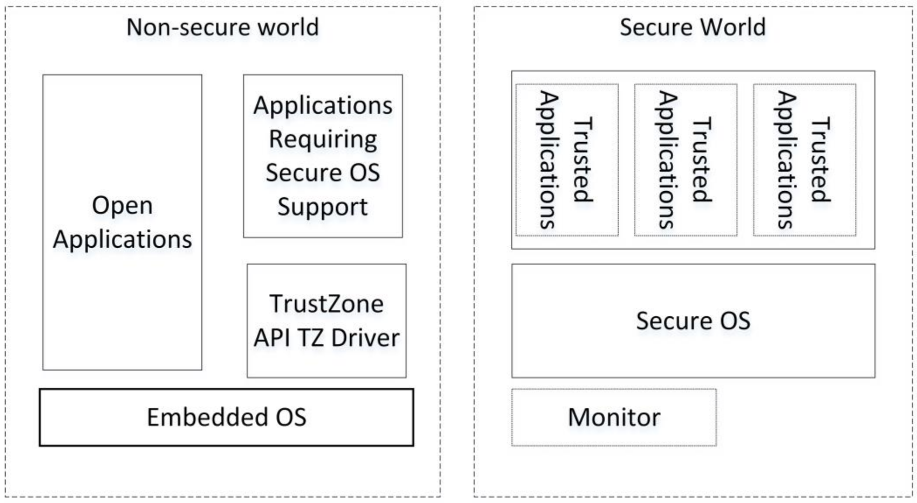


Figure 2.1

a. Secure operating system (The highest level of software architecture)

A dedicated operating system in the secure world is a complex, but powerful, design. It can simulate concurrent execution of multiple independent Secure world applications, run-time download of new security applications, and Secure world tasks that are completely independent of the Normal world environment.

The most extreme version of these designs closely resembles the software stacks that would be seen in a SoC with two separate physical processors in an Asymmetric Multi-Processing (AMP) configuration. The software running on each virtual processor is a standalone operating system, and each world uses hardware interrupts to preempt the currently running world and acquire processor time. A tightly integrated design, which uses a communications protocol that associates Secure world tasks with the Normal world thread that requested them, can provide many of the benefits of a Symmetric Multi-Processing (SMP) design. In these designs a Secure world application could, for example, inherit the priority of the Normal world task that it is assisting. This would enable some form of soft real-time response for media applications.

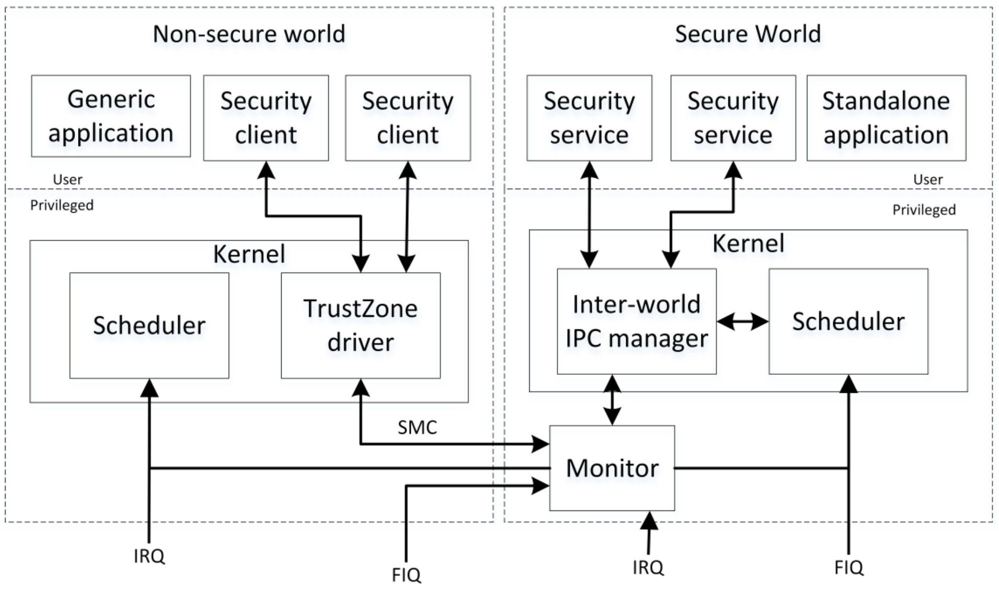


Figure 2.2

b. Synchronous library

Many use cases don't need the complexity of a secure world operating system. We have a simple way to solve the problem. In the secure world, a simple library of code handling a task at one time is sufficient for many applications. The code library is scheduled and managed using software calls from the Normal world operating system. The secure world in these systems is a slave to the normal world and cannot operate independently, but can consequently have a much lower level of complexity.

2.2 SMC

A *Secure Monitor Call* (SMC) is used to enter the Secure Monitor mode and perform a Secure Monitor kernel service call. SMC instruction is a specific command. It is similar to software interrupt instruction (SWI), which can be used to call a monitor routine.

This is the simplest way to enter the secure world from the normal world. When the users in the normal world want to access to service of the secure world, they first need to enter the privileged mode in the normal world. In this mode with SMC instruction, the processor will go into monitor mode. Monitor mode copies context of the normal world and enters into privileged mode of the secure world. As shown in processing mechanism (Figure 1.1), operating environment at this time is in a execution environment of the secure world. After entering into user mode of the secure world, it will execute appropriate security services. The reason why we should separate the user mode and privilege mode in the secure world is that privileged mode execution environment usually is a system-level while the security services of user mode is application-level.

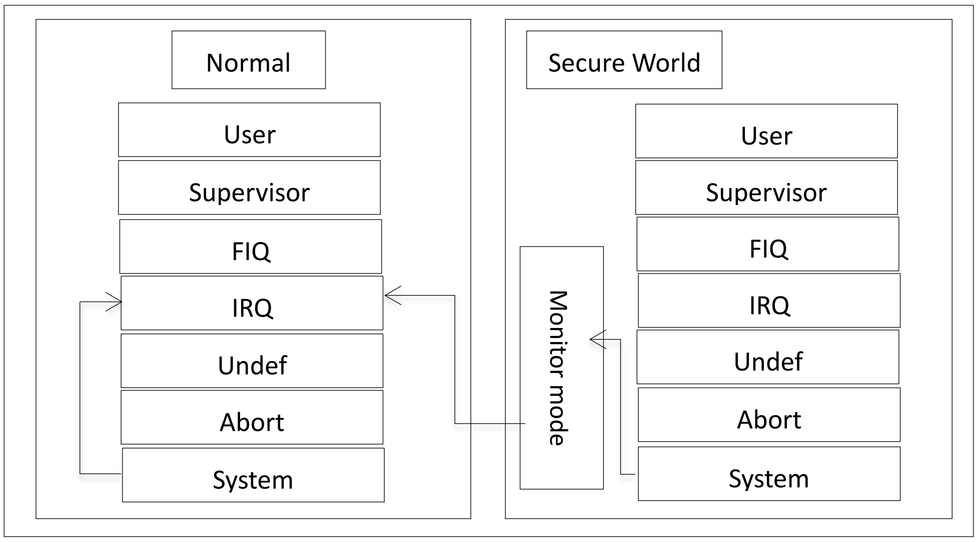


Figure 2.3

There is a secure configuration register (SCR) in a ARM processor with TrustZone security extensions. In the SCR, there exists a NS bit, which indicates the current state of the system. If NS is equal to 0, the current system is in secure state. If NS is equal to 1, the current system is in a non-secure state. When the system is in monitor mode, regardless of whether the NS is equal to 0, users can access the resources in the security environment. NS bit is the TrustZone critical extensions to the system. The user mode and the privilege mode of system is independent of the security status of the system, which means that user's programs can also be run in secure mode and an application running in the privileged mode may also be in the non-secure state. NS bit can only be changed by a software running in a privileged mode of secure state. And the system cannot access the SCR registers when it is in a non-secure state. Security environment software runs in a secure state of the environment, while general environment software runs in non-secure state. Therefore, when users switch the execution environment, they need to change the security status of the system at the same time. The following diagram is about SCR register format (Figure 2.4):

	AW	FW	EA	FIQ	IRQ	NS

Figure 2.4

According to available information above, based on the design of this processor mode with TrustZone, there are three main mechanisms switching from the normal world to the secure world through the monitor:

External Suspension includes external aborts of prefetching and data. External stops are at the time of accessing the storage system but will not be detected by the MMU. The exception generally occurs when accessing the resource from the normal environment to the security environment.

Disruption includes the IRQ and FIQ interruption. In three methods of entering a monitor mode we have talked above, the first method entering the monitor mode is unconditional. Both of the rest depend on configuration of SCR register. SCR register fields associated with the monitor mode is defined as follows:

1. EA, the default is 0, represents processor goes into suspended mode when external abort is happened. If it is equal to 1, it represents processor goes into monitor mode when external abort is happened;
2. IRQ, the default is 0, indicates that when IRQ interrupt is happened, processor enters into suspend mode. If it is equal to 1, it indicates the processor enter into the monitor model;
3. FIQ, the default is 0, indicates that when FIQ interrupt is happened, processor enters into suspend mode. If it is equal to 1, the processor goes into monitor mode.

The three exceptions above will trigger monitor mode exception handler. Monitor function is defined by software developers, but switching the execution environment is conducted through the monitor. If the software in the secure environment switches the NS bit to 1, the system directly goes into non-security state, which allows to see lines of instructions as well as data in the registers. If these instructions and data are sensitive information, it would pose a security threat to the system. Therefore, normally only the monitor can be directly modified the NS bit. Based on the analysis above, we decided to adopt the design, which is consisted of two virtual processor cores through the implementation of SMC command switching modes.

IRQ is recommended as a general source of environmental disruptions by ARM and FIQ is treated as interrupted source in the security environment. Most operating systems use IRQ as interrupted source. So using FIQ as interrupted source of security means the general environment is changed least. If the interruption occurred in the corresponding execution environment, it does not need to switch the execution environment. Or it can get into the monitor mode and monitor switches the execution environment. Under normal circumstances, when executing the monitor code, the system should turn off the break.

Mode	Mode category	NS bit=1	NS bit=0
User	User	Not safe	safe
FIQ	privilege	Not safe	safe
IRQ	privilege	Not safe	safe
Abort	privilege	Not safe	safe
Undef	privilege	Not safe	safe
System	privilege	Not safe	safe
Supervisor	privilege	Not safe	safe

Figure 2.5

The three exceptions above will trigger monitor mode exception handler. Monitor function is defined by software developers, but switching the execution environment is conducted through the monitor. If the software in the secure environment switch the NS bit to 1, the system directly goes into non-security state, which allows to see lines of instructions as well as data in the registers. If these instructions and data are sensitive information, it would pose a security threat to the system. Therefore, normally only the monitor can be directly modified the NS bit. Based on the analysis above, we decided to adopt the design, which is consisted of two virtual processor cores through the implementation of SMC command switching modes.

2.3 Mechanism

a. Figure 2.6 shows how to switch between Normal world and Secure world via Monitor mode and SMC mechanism.

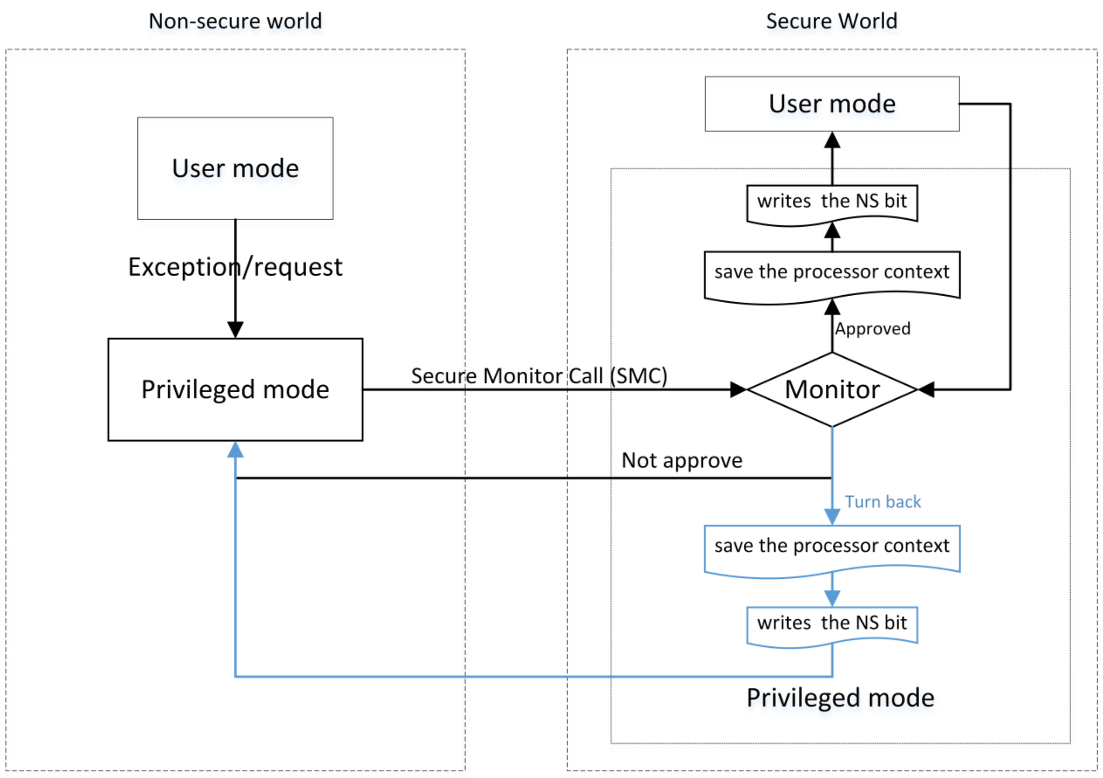


Figure 2.6

When a User process wants to request a change from one world to the other it must first execute a SVC instruction. This changes the processor to a privileged mode where the Supervisor call handler processes the SVC and executes a SMC. Then the Secure Monitor determines if a change from one world to the other is valid. If so, it writes to the NS bit to change the world in operation. In addition, Secure Monitor saves the processor context, which includes register banks. When the Secure Monitor transfers control from one world to the other it must save the processor context.

b. initialize TrustZone

According to information online, we should pay attention to the initialization of TrustZone. In addition to running time security, we should also defend the attacks which encounter when the TrustZone is booting. As a result, the initialization of TrustZone should be re-booted in a specific order so that no chances are left for hacker to attack. The order of initialization is as follow (figure 2.7)

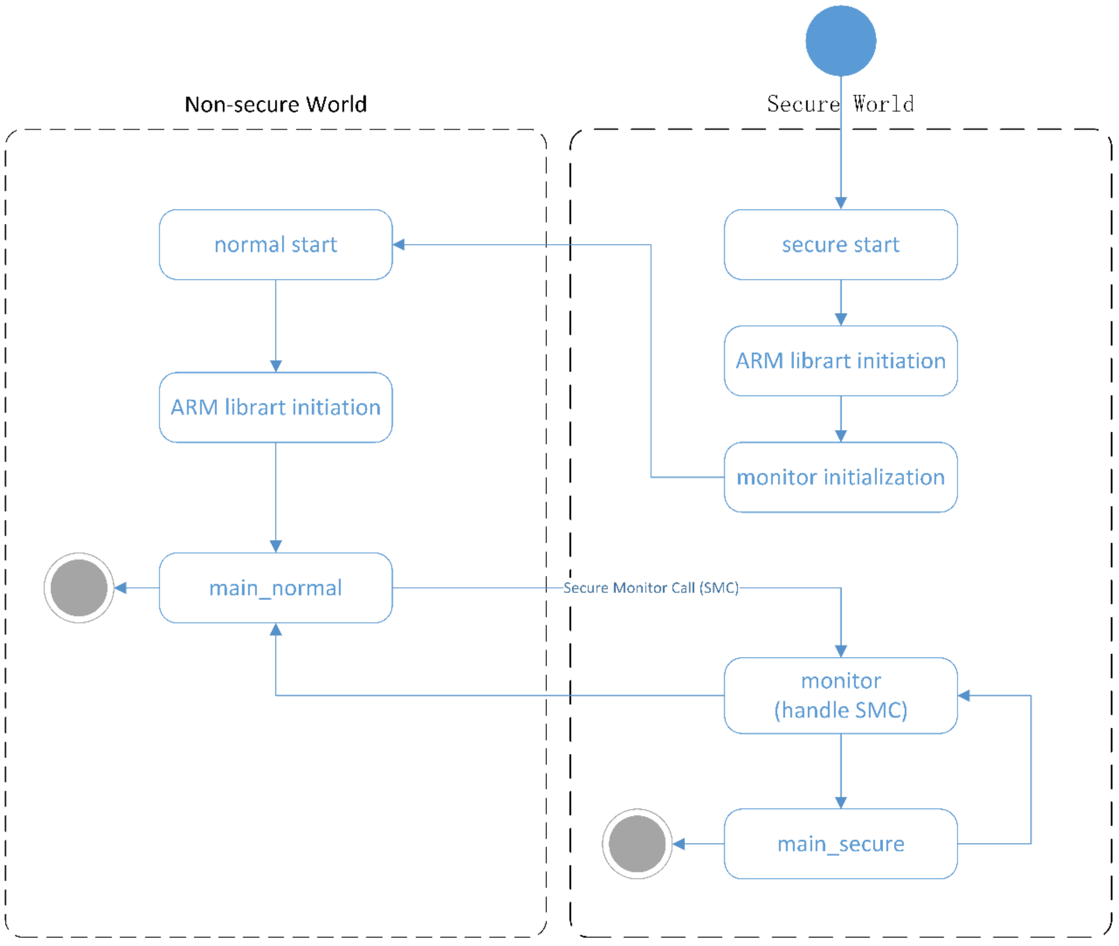


Figure 2.7

In an initialization, the steps are as follow:

1. Initialize the Secure World, which including ARM library, cache (in Global) and Monitor initialization.
2. after secure start, monitor mode will call NS world to switch to it and let normal world initializes.
3. the library of normal world will be initialized as well as the caches belonged to it.
4. after Normal World initialized, it executes SMC to switch to Secure world.
5. monitor perform context witch from Normal World to Secure World (saves the processor context and changes the NS bit)

3. TrustZone study in DS-5

We try to implement the basic function of trust zone and understand it better through the exploring of the real example. Because we don't have any ARM hardware to do the test, we used some Integrated Development Environment (IDE) to do the study before simulate with C++ programs. We choose to use ARM DS-5 to do the study. DS-5 is a professional software development solution for Linux-based systems and bare-metal embedded systems, covering all stages in development from boot code and

kernel porting to application and bare-metal debugging including performance analysis. Eclipse for DS-5 is an IDE with the compilation and debug technology of the ARM® tools. To use the full function of DS-5, we signed for a 30-day evaluation. Through eclipse we can track with the Registers state, memory and program steps as well as the result in the same time.
Through the eclipse, we tested and modified a basic example of trust zone to help us understand it. The example is targeted at Versatile Express Cortex-A9x1 FVP model. The example shows the switch between the Secure world and Non-secure world linked via a Secure Monitor. Although there are many functions of a real Secure Monitor, the example only provide a partial context switch through Supervisor Call (SVC) mode and only save/restore the SVC mode register, and the Trust Zone Protection Controller (TZPC) is configured to allow non-secure access to all memory and peripherals.
The program are combined with several documents. The main.c documents of Secure world and Non-secure world, the assembly language documents of startup the Secure world and Non-secure world and Monitor are the studied and modified by us in order to understand the program.
The program execution flow is the same as we discussed before, here are the detailed steps and the description.

3.1 Initialization

The program starts from Secure world, before execution the main function, it will initialize Secure world, Monitor and Non-secure world in an order.

3.1.1 Initialize Secure world

Program first initializes the Secure world through executing startup_secure.s.

a. The initialization of Secure world include setting up SVC stack, invalidating caches, TLBs, and setting up page tables, entry for TZPC, enabling MMU.

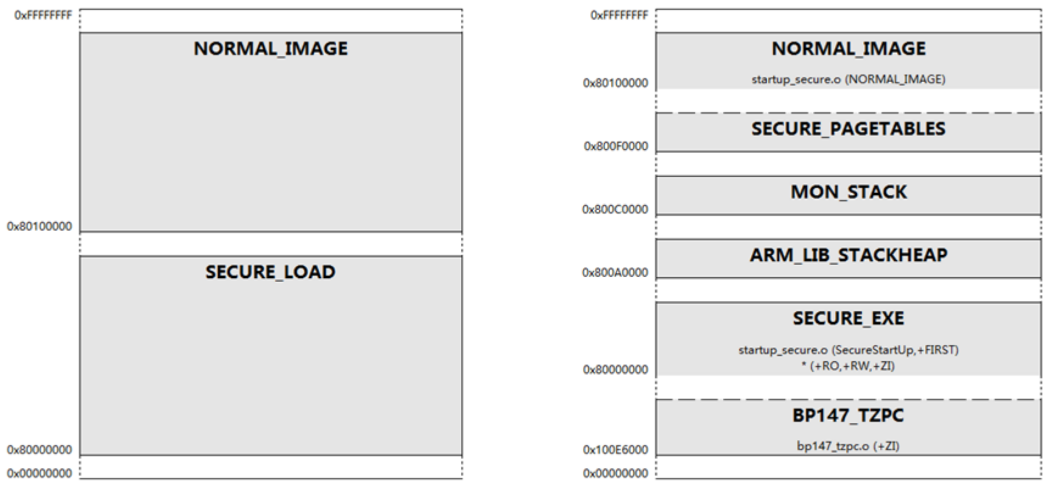
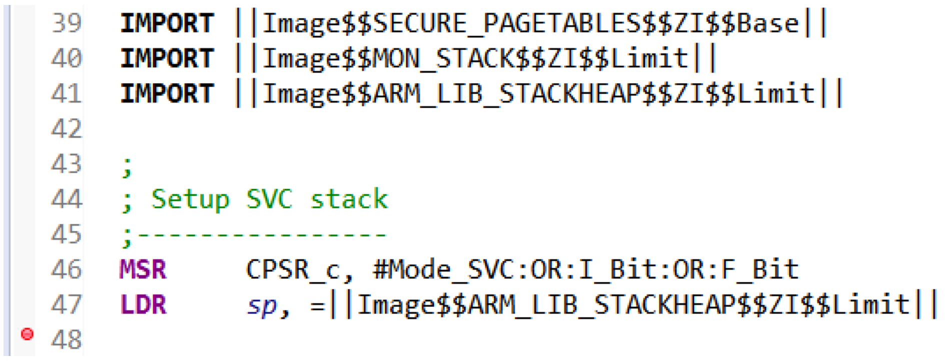


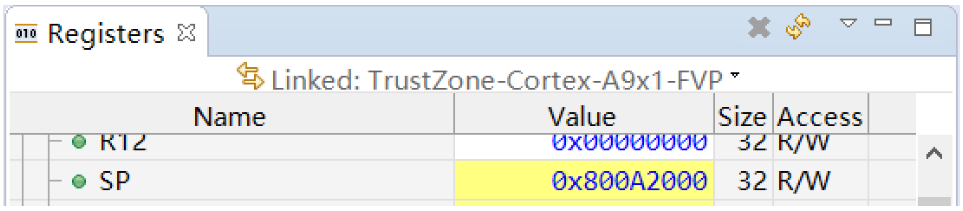
Figure 3.1

The program using linker pre-define symbols to set up address. Every parts of the components will be the same as image shows.



```
39  IMPORT  ||Image$$SECURE_PAGETABLES$$ZI$$Base||
40  IMPORT  ||Image$$MON_STACK$$ZI$$Limit||
41  IMPORT  ||Image$$ARM_LIB_STACKHEAP$$ZI$$Limit||
42
43  ;
44  ; Setup SVC stack
45  ;----------------
46  MSR     CPSR_c, #Mode_SVC:OR:I_Bit:OR:F_Bit
47  LDR     sp, =||Image$$ARM_LIB_STACKHEAP$$ZI$$Limit||
48
```

Figure 3.2



Registers

Linked: TrustZone-Cortex-A9x1-FVP

Name	Value	Size	Access
R12	0x00000000	32	R/W
SP	0x800A2000	32	R/W

Figure 3.3

For example, when set up the SVC stack, after execution to 48, we can find out the sp register get a value of 0x800A2000. When we check about the code of the image, it is "ARM_LIB_STACKHEAP   0x800A0000 EMPTY 0x2000   {}".

a. After this, the program will branch to C library run-time initialization in __main.

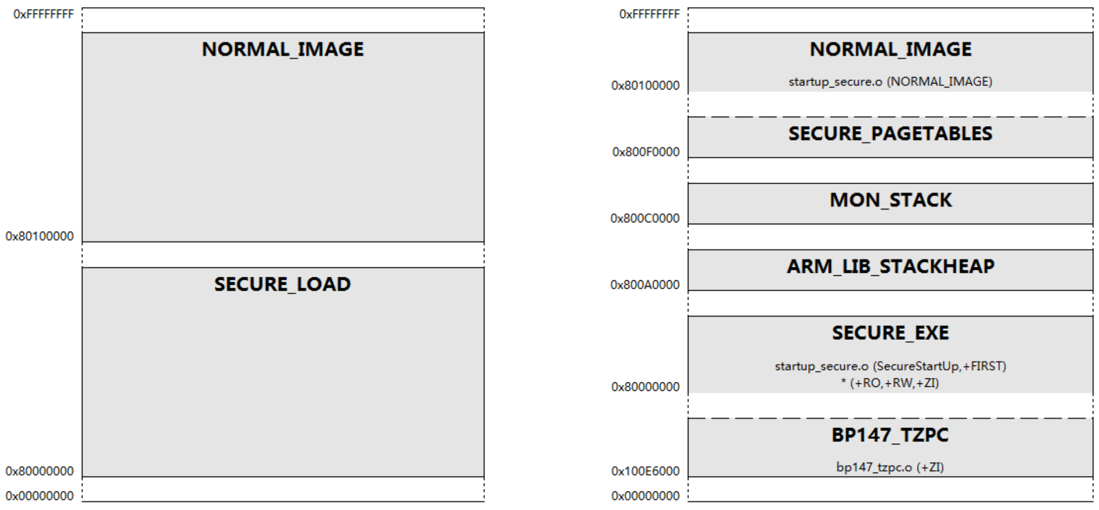
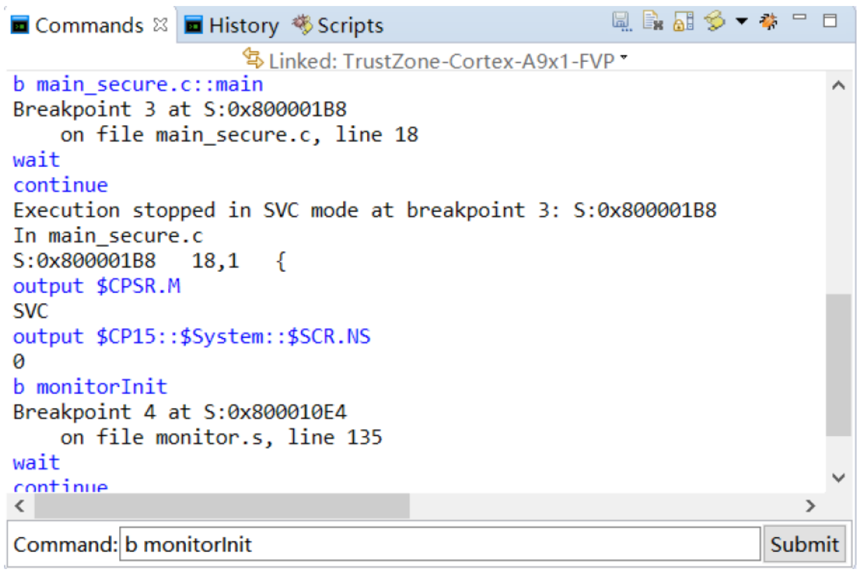


Figure 3.4

After initializing the Secure world, program will execute main() function of Secure world. We can check at the M bit in the CPSR and NS bit in the Secure Configuration Register (SCR) (Figure 3.4), knowing that know the processor is in SVC mode in Secure world.



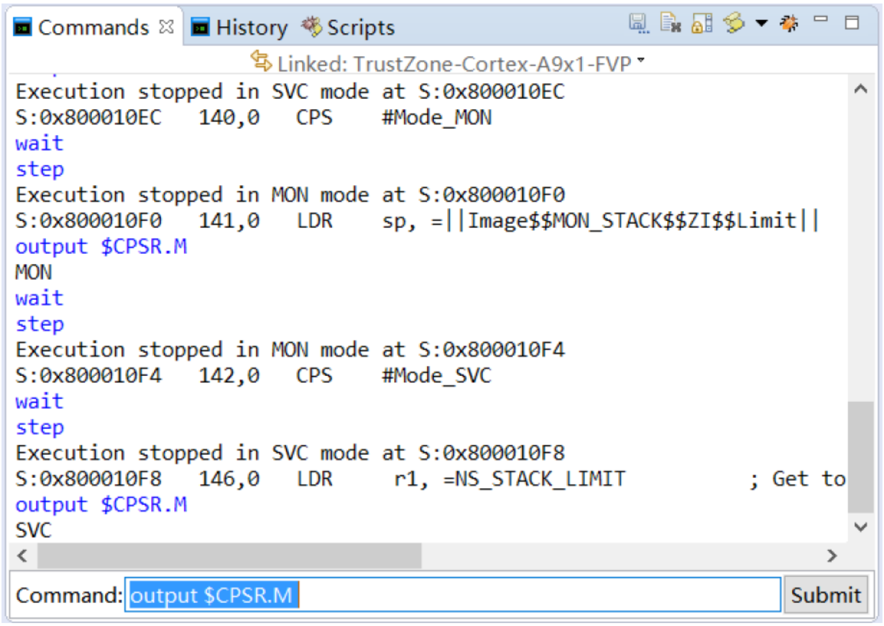
```
b main_secure.c::main
Breakpoint 3 at S:0x800001B8
    on file main_secure.c, line 18
wait
continue
Execution stopped in SVC mode at breakpoint 3: S:0x800001B8
In main_secure.c
S:0x800001B8   18,1   {
output $CPSR.M
SVC
output $CP15::$System::$SCR.NS
0
b monitorInit
Breakpoint 4 at S:0x800010E4
    on file monitor.s, line 135
wait
continue
```

```
Command: b monitorInit                                        Submit
```

Figure 3.5

3.2 Initialize Secure Monitor

a. In the main() function, the program will call the monitorInit() function to install monitor. The program will jump into monitorInit() in monitor.s and install Secure Monitor. The Monitor initialization will install the Monitor's vector table, initialize the Monitor mode stack pointer, create and save a dummy Normal world state that will be used for the first entry to the Normal world before returning to the Secure world caller. During this execution, the mode will change from SVC to MON and change back again, which reflected in the CPSR.



```
Execution stopped in SVC mode at S:0x800010EC
S:0x800010EC    140,0    CPS     #Mode_MON
wait
step
Execution stopped in MON mode at S:0x800010F0
S:0x800010F0    141,0    LDR     sp, =||Image$$MON_STACK$$ZI$$Limit||
output $CPSR.M
MON
wait
step
Execution stopped in MON mode at S:0x800010F4
S:0x800010F4    142,0    CPS     #Mode_SVC
wait
step
Execution stopped in SVC mode at S:0x800010F8
S:0x800010F8    146,0    LDR      r1, =NS_STACK_LIMIT          ; Get to
output $CPSR.M
SVC
```

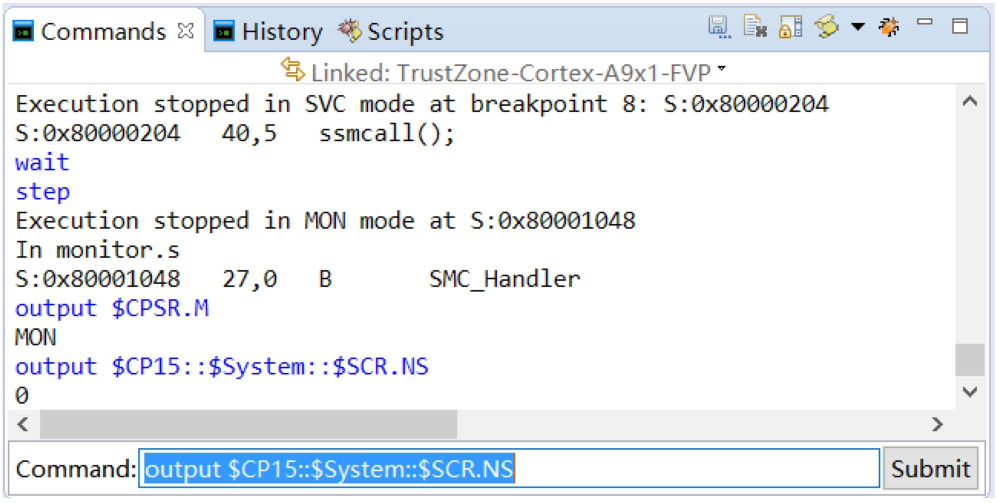
```
Command: output $CPSR.M                                       Submit
```

Figure 3.6

b. After initialization, the monitor can be called through several ways. In the program, we call it through a SMC exception, triggered into SMC Handler. The SMC Handler in Secure Monitor will determine which world we come from through checking the NS bit. Than save the current world address from the point or, general purpose registers, SPSR, LR and SP; restore address of the other world to the pointer, restore other world's registers, SPSR and LR. In the end, before stepping the exception return instruction "MOVS pc, lr", the NS bit will be reset to show the correct world state and toggled.

3.3 Initialize Non-secure world

a. After returning to the Secure world, the main() will print the first "Hello from Secure world i=0", then execute an SMC instruction via ssmcall() to switch back to the Monitor. The ssmcall() triggers an SMC exception, landing at the SMC entry in the Monitor's vector table to the SMC handler, and the mode change from SVC to MON reflected in the CPSR.

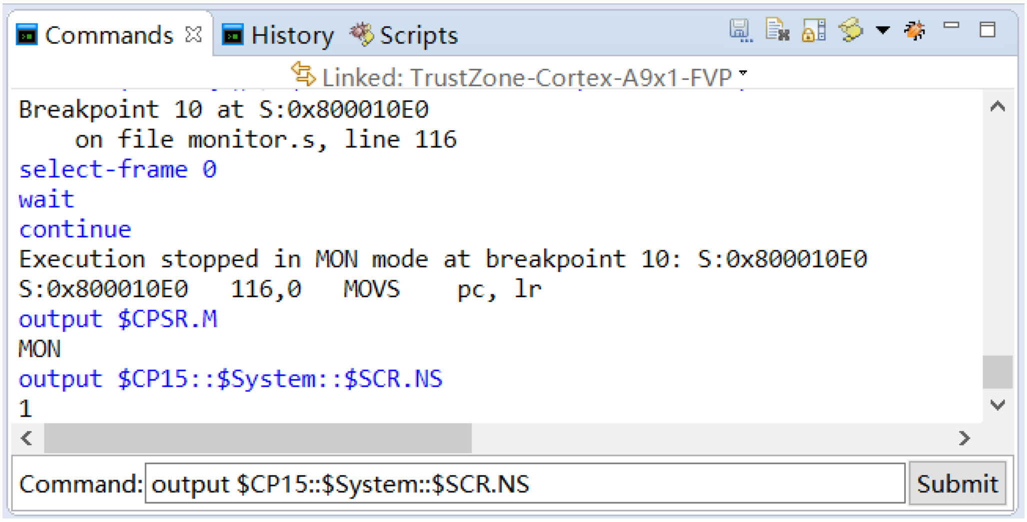


```
Execution stopped in SVC mode at breakpoint 8: S:0x80000204
S:0x80000204   40,5   ssmcall();
wait
step
Execution stopped in MON mode at S:0x80001048
In monitor.s
S:0x80001048   27,0   B       SMC_Handler
output $CPSR.M
MON
output $CP15::$System::$SCR.NS
0
```

Command: output $CP15::$System::$SCR.NS

Figure 3.7

Before stepping the exception return instruction, the NS bit is reset to 1 because the program will switch from Secure world to Non-secure world.



```
Breakpoint 10 at S:0x800010E0
    on file monitor.s, line 116
select-frame 0
wait
continue
Execution stopped in MON mode at breakpoint 10: S:0x800010E0
S:0x800010E0   116,0   MOVS    pc, lr
output $CPSR.M
MON
output $CP15::$System::$SCR.NS
1
```

Command: output $CP15::$System::$SCR.NS

Figure 3.7

b. The program jumps from monitor to Non-secure world's normalStart routine. In the non-secure world initialization, the program will put all the CPUs except core 0 to sleep, set up stacks, invalidating the caches and TLBs, set up page tables, enable the MMU, set the Vector Base Address Register and then branch to C library run-time initialization in __main.

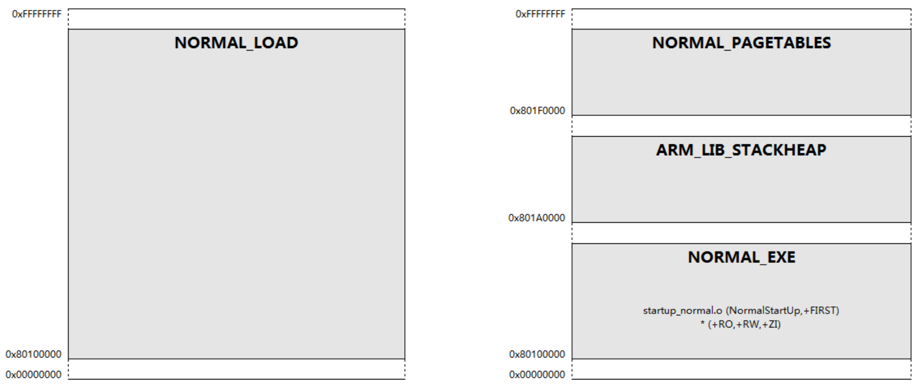
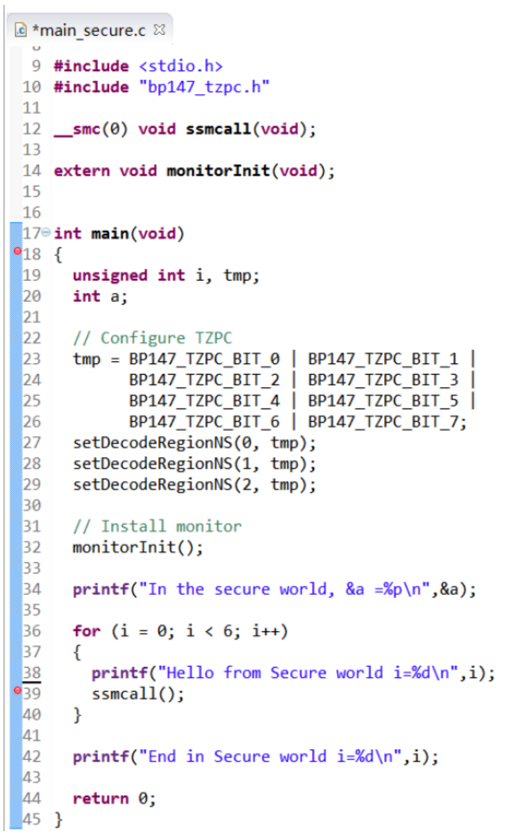


Figure 3.8

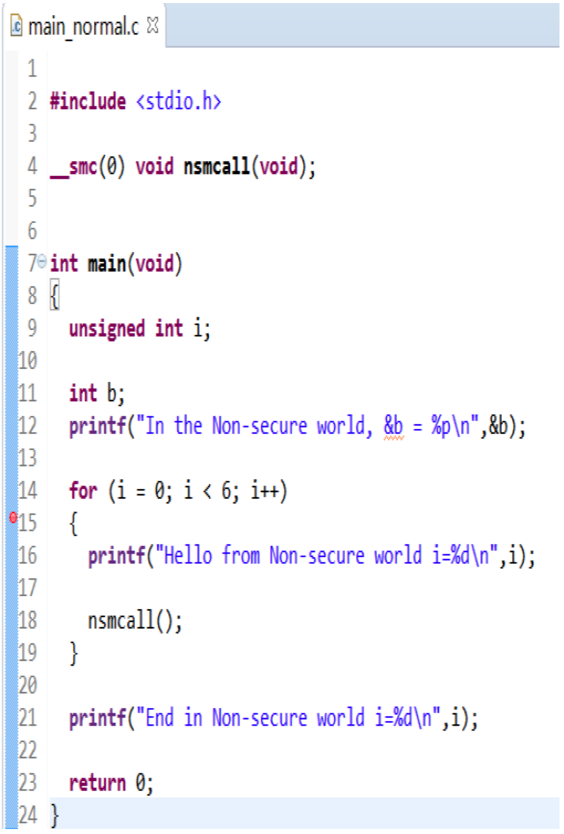
3.4Main program

Figure 3.9 is the main() function of Secure world and Non-secure world.



*main_secure.c

```c
#include <stdio.h>
#include "bp147_tzpc.h"

__smc(0) void ssmcall(void);

extern void monitorInit(void);


int main(void)
{
  unsigned int i, tmp;
  int a;

  // Configure TZPC
  tmp = BP147_TZPC_BIT_0 | BP147_TZPC_BIT_1 |
        BP147_TZPC_BIT_2 | BP147_TZPC_BIT_3 |
        BP147_TZPC_BIT_4 | BP147_TZPC_BIT_5 |
        BP147_TZPC_BIT_6 | BP147_TZPC_BIT_7;
  setDecodeRegionNS(0, tmp);
  setDecodeRegionNS(1, tmp);
  setDecodeRegionNS(2, tmp);

  // Install monitor
  monitorInit();

  printf("In the secure world, &a =%p\n",&a);

  for (i = 0; i < 6; i++)
  {
    printf("Hello from Secure world i=%d\n",i);
    ssmcall();
  }

  printf("End in Secure world i=%d\n",i);

  return 0;
}
```



main_normal.c

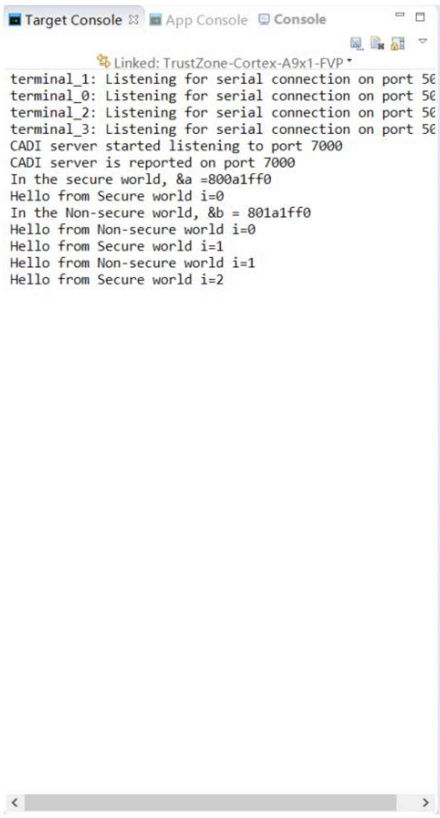
```c

#include <stdio.h>

__smc(0) void nsmcall(void);


int main(void)
{
  unsigned int i;

  int b;
  printf("In the Non-secure world, &b = %p\n",&b);

  for (i = 0; i < 6; i++)
  {
    printf("Hello from Non-secure world i=%d\n",i);

    nsmcall();
  }

  printf("End in Non-secure world i=%d\n",i);

  return 0;
}
```

Figure 3.9(a) **Figure 3.9(b)**

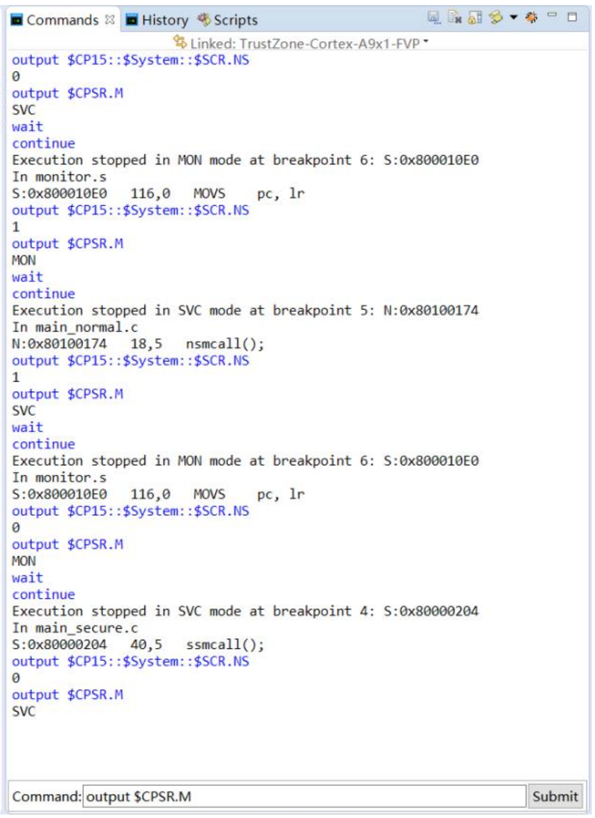
After initializing Secure world, Secure Monitor and Non-secure world, the program are now executing in the main() function in the Non-secure world. When the program execute in the loop, it will print first “Hello from Non-secure world i=0” and execute an SMC instruction via nsmcall() to switch into the Monitor.

In the Secure Monitor, the program will execute SMC handler, the same as we described in 3.9 b). The mode changes between SVC and MON, the NS bit changes in different world in every time of the loop.

In our program, each time steps into the Securer Monitor, it will lead to a switch from one world to the other. First of all, the program in loop 1 in the Secure world, after printing the sentence, the monitor switches the program into the Non-secure world main() function. In the Non-secure world main(), the program steps into another loop. In this loop 1, after printing the sentence of the world, the monitor will switch into Secure world, right after the SMC exception, so the program will continue the loop 2 in the Secure world and so on. Figure 3.10 (a) shows parts of the results of the program during the loop; (b) shows the CPSR register and NS bit register’s state change as program in Secure world—Secure Monitor—Non-secure world—Secure Monitor—Secure world, corresponding to result “i=1” and “i=2” in figure 3.10(a).



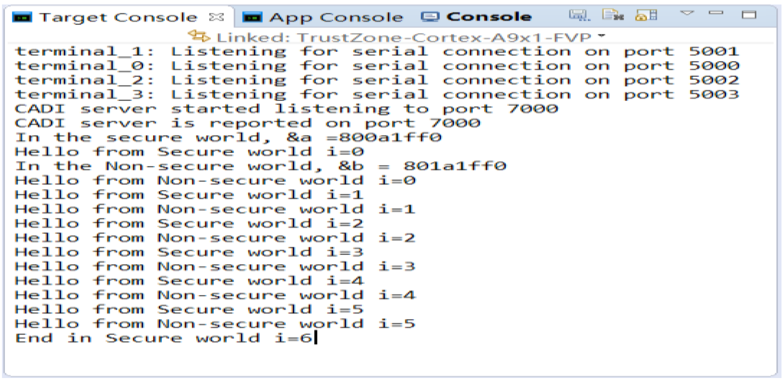
```
terminal_1: Listening for serial connection on port 5
terminal_0: Listening for serial connection on port 5
terminal_2: Listening for serial connection on port 5
terminal_3: Listening for serial connection on port 5
CADI server started listening to port 7000
CADI server is reported on port 7000
In the secure world, &a =800a1ff0
Hello from Secure world i=0
In the Non-secure world, &b = 801a1ff0
Hello from Non-secure world i=0
Hello from Secure world i=1
Hello from Non-secure world i=1
Hello from Secure world i=2
```



```
output $CP15::$System::$SCR.NS
0
output $CPSR.M
SVC
wait
continue
Execution stopped in MON mode at breakpoint 6: S:0x800010E0
In monitor.s
S:0x800010E0   116,0   MOVS    pc, lr
output $CP15::$System::$SCR.NS
1
output $CPSR.M
MON
wait
continue
Execution stopped in SVC mode at breakpoint 5: N:0x80100174
In main_normal.c
N:0x80100174   18,5   nsmcall();
output $CP15::$System::$SCR.NS
1
output $CPSR.M
SVC
wait
continue
Execution stopped in MON mode at breakpoint 6: S:0x800010E0
In monitor.s
S:0x800010E0   116,0   MOVS    pc, lr
output $CP15::$System::$SCR.NS
0
output $CPSR.M
MON
wait
continue
Execution stopped in SVC mode at breakpoint 4: S:0x80000204
In main_secure.c
S:0x80000204   40,5   ssmcall();
output $CP15::$System::$SCR.NS
0
output $CPSR.M
SVC
```

Figure 3.10 (a) **Figure 3.10 (b)**

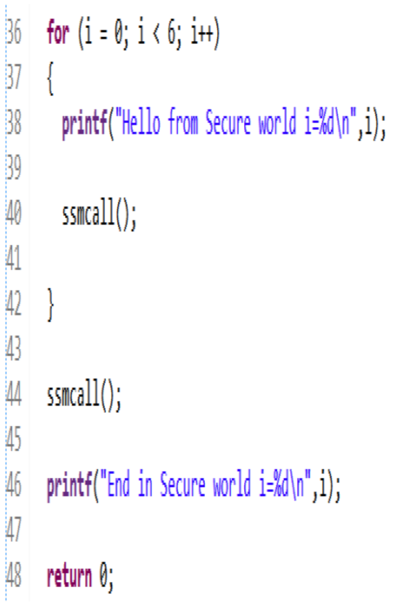
In the final loop, after switching back from Non-secure world in to the secure world, the loop will end. Because we designed to have SMC exception only in the loop, the program will not call monitor after the loop, and the program will end in the Secure world. The result shows as figure 3.11.



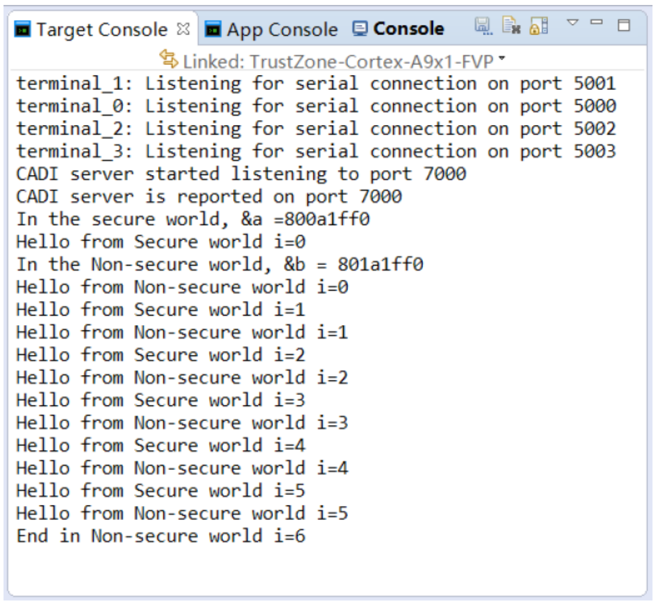
```
terminal_1: Listening for serial connection on port 5001
terminal_0: Listening for serial connection on port 5000
terminal_2: Listening for serial connection on port 5002
terminal_3: Listening for serial connection on port 5003
CADI server started listening to port 7000
CADI server is reported on port 7000
In the secure world, &a =800a1ff0
Hello from Secure world i=0
In the Non-secure world, &b = 801a1ff0
Hello from Non-secure world i=0
Hello from Secure world i=1
Hello from Non-secure world i=1
Hello from Secure world i=2
Hello from Non-secure world i=2
Hello from Secure world i=3
Hello from Non-secure world i=3
Hello from Secure world i=4
Hello from Non-secure world i=4
Hello from Secure world i=5
Hello from Non-secure world i=5
End in Secure world i=6
```

Figure 3.11

If the loop of Secure world is more than Non-secure world, or after the loop, the Secure world execute ssmcall() one more time. The monitor will switch into Non-secure world, and if there is no more trigger (nsmcall()), the program will end in Non-secure world. The change like figure 3.12(a) can lead to a result as 3.12(b).



```
for (i = 0; i < 6; i++)
{
  printf("Hello from Secure world i=%d\n",i);

  ssmcall();

}

ssmcall();

printf("End in Secure world i=%d\n",i);

return 0;
```



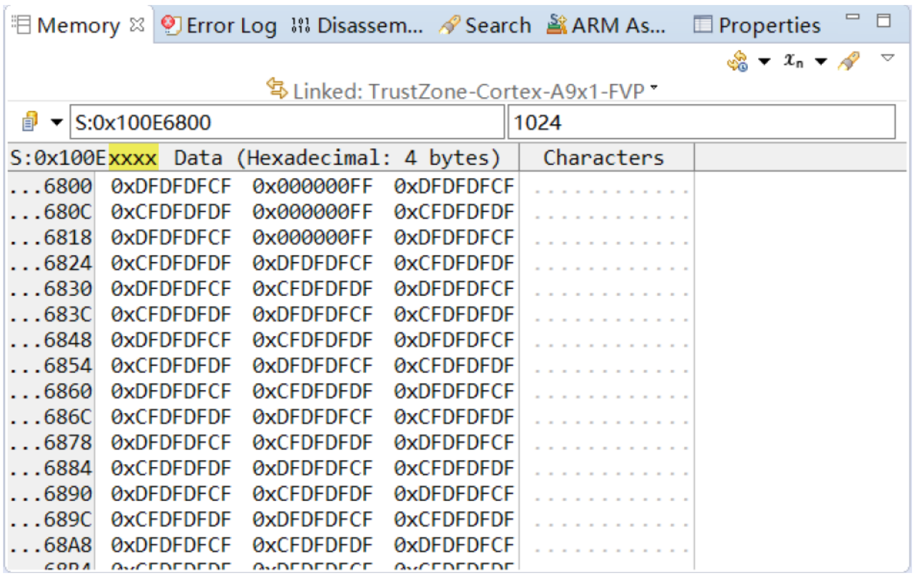
```
terminal_1: Listening for serial connection on port 5001
terminal_0: Listening for serial connection on port 5000
terminal_2: Listening for serial connection on port 5002
terminal_3: Listening for serial connection on port 5003
CADI server started listening to port 7000
CADI server is reported on port 7000
In the secure world, &a =800a1ff0
Hello from Secure world i=0
In the Non-secure world, &b = 801a1ff0
Hello from Non-secure world i=0
Hello from Secure world i=1
Hello from Non-secure world i=1
Hello from Secure world i=2
Hello from Non-secure world i=2
Hello from Secure world i=3
Hello from Non-secure world i=3
Hello from Secure world i=4
Hello from Non-secure world i=4
Hello from Secure world i=5
Hello from Non-secure world i=5
End in Non-secure world i=6
```

Figure 3.12 (a) **Figure 3.12 (b)**

3.5 Conclusion

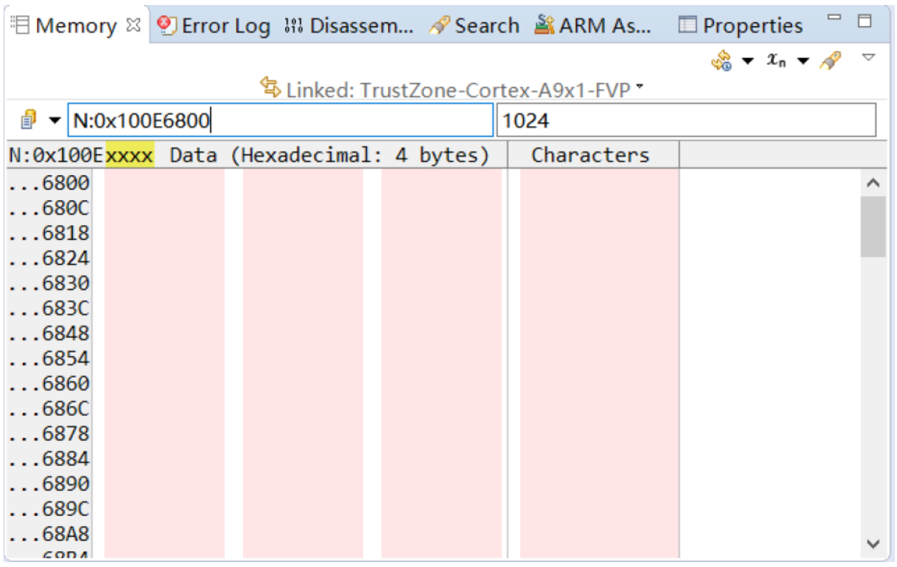
a. Through the detailed study of this program, we can understand more clearly about the trust-zone design principle. Find out the massive assemble language work beyond the higher level language. With the recording of the registers data change, we can track how does the monitor save and restore the pointer of the two world, thus we can track the way to switch the world returning to the SMC exceptions.

b. This is a simple program trying to show the trust-zone work flow, so there are also several cons. In a real system, the Trust Zone Protection Controller (TZPC) can be is configured to allow/forbid secure/non-secure access to memory and peripherals. This program is configured to allow non-secure access to all memory. However, the TZPC itself is only accessible in secure world. We can demonstrate this through Memory view, the figure 5.1 shows the memory view at address S: 0x100E6800 (S: meaning Secure access). The TZPC is based at 0x100E6000 but its status/control registers start at offset 0x800. The figure 5.2 shows the memory view at address N: 0x100E6800 (N: meaning Non-Secure access). This address region will abort because the TZPC is not accessible from Non-Secure state.The program are designed to show the switch of the two worlds, it doesn’t have the useable interface to exchange data and instructions between the two worlds.



```
S:0x100E6800                1024
S:0x100Exxxx  Data (Hexadecimal: 4 bytes)       Characters
...6800  0xDFDFDFCF  0x000000FF  0xDFDFDFCF   ............
...680C  0xCFDFDFDF  0x000000FF  0xCFDFDFDF   ............
...6818  0xDFDFDFCF  0x000000FF  0xDFDFDFCF   ............
...6824  0xCFDFDFDF  0xDFDFDFCF  0xCFDFDFDF   ............
...6830  0xDFDFDFCF  0xCFDFDFDF  0xDFDFDFCF   ............
...683C  0xCFDFDFDF  0xDFDFDFCF  0xCFDFDFDF   ............
...6848  0xDFDFDFCF  0xCFDFDFDF  0xDFDFDFCF   ............
...6854  0xCFDFDFDF  0xDFDFDFCF  0xCFDFDFDF   ............
...6860  0xDFDFDFCF  0xCFDFDFDF  0xDFDFDFCF   ............
...686C  0xCFDFDFDF  0xDFDFDFCF  0xCFDFDFDF   ............
...6878  0xDFDFDFCF  0xCFDFDFDF  0xDFDFDFCF   ............
...6884  0xCFDFDFDF  0xDFDFDFCF  0xCFDFDFDF   ............
...6890  0xDFDFDFCF  0xCFDFDFDF  0xDFDFDFCF   ............
...689C  0xCFDFDFDF  0xDFDFDFCF  0xCFDFDFDF   ............
...68A8  0xDFDFDFCF  0xCFDFDFDF  0xDFDFDFCF   ............
```

Figure 5.1



```
N:0x100E6800                1024
N:0x100Exxxx  Data (Hexadecimal: 4 bytes)       Characters
...6800
...680C
...6818
...6824
...6830
...683C
...6848
...6854
...6860
...686C
...6878
...6884
...6890
...689C
...68A8
```

Figure 5.2

4. Simulating the Mechanism of Trustzone (C++)

4.1 Mechanism of C++ Program Simulating Trustzone

According to the mechanism of the Trustzone, there are two worlds: one is named the normal world, the other is the secure world. It uses the monitor to switch between the normal world and the secure world. If checking an interruption in the normal world. (For example, if there is no enough resource in the normal to continue the operation safely. And the secure world has enough big space to continue the operation), it will use the instruction, SMC, to jump into the monitor which is in the secure world. At the same time, the SMC instruction will copy the data from the normal world to the monitor. And the monitor will transfer the data into secure world. The secure world will use its resource to continue the operation.
For simulating the mechanism of Trustzone by using a computing example, we use the C++ language to build a program. In the program, we just can compute and show the lowest four digits in the normal world due to limited resource. If the number of digits of figures is larger than 4, it cannot calculate the numbers in the normal world because the normal world has the limited space whose length is 4. The program will produce an interruption in the normal world due to overflow. When checking the interruption, the SMC will be activated and copy the data into the monitor in the secure world. If the monitor receives the data, it will give the data to the secure world and the data will use the larger space in the secure world to continue the computing. The result will return to the normal world through the monitor using the SMC instruction.

4.2 Development Platform

We use the visual studio 2013 to build our program based on the WIN7 OS.

4.3 Development

4.3.1 Design of The Program

According to the principle of the Trustzone, we create 2 classes, one is Trustzone, the other is SecureWorld. And we will have 3 .cpp files to implement the method in the .h files. Now we draw the UML chart for our program:

SecureWorld
- longinteger[maxdigits] : short - short inputvalue[maxdigits] : short
+ maxdigits = 30: static const int <constructor> + TrustzonesecureWorld () <constructor> + TrustzonesecureWorld (& op:Trustzone) <constructor> + TrustzonesecureWorld (v : long) +secureWorld (&op1:Trustzone, & op2:Trustzone) : int +operator+ (&op2 :TrustzonesecureWorld) : TrustzonesecureWorld +OverflowWarning() :void +Monitor(&op1 :Trustzone , &op2 :Trustzone) : int +ChangetoInt(&op :TrustzonesecureWorld) : int operator<<(&output: ostream , &num:TrustzonesecureWorld): friend ostream&

Trustzone
-inputvalue: long
+ digits = 4: static const int +integer[digits]: short <constructor> +Trustzone(long = 0); +operator+(&op2:Trustzone) :Trustzone +min(a:int , b: int) : int + normalWorld(&op1:Trustzone, &op2:Trustzone): Trustzone +SMC(&op1:Trustzone, &op2:Trustzone) :int +setvalue(v:long):void +getvalue():int +OverflowWarning():void +ChangetoInt(&op:Trustzone):int + CannotContain() : void operator<<(&output: ostream , &num:Trustzone): friend ostream&

a. Class SecureWorld:

In the class, maxdigits is equal to 30, so we use the arrays, inputvalue and longinteger, to represent the space which the secure world can use. The function of secureWorld is used to implement the action of secure world. And the function operator+ is overloaded the function of addition operation. So the secure world can use the overloaded addition function to compute big data. The function of Monitor is used to check if there is a SMC instruction between the normal world and secure world. If there is a SMC instruction, the data will be transferred between the two worlds through monitor function. The number of ChangetoInt function is used to change the results computed in the secure world into int type. Overflow function is used to show if the result overflows in the secure world because the length of the space in secure world is 30, even though it is larger than the space in normal world. We overload the operator<< function to implement the output of the result computed in the normal world.

b. Class Trustzone:

In the class, there is a long type variable (inputvalue) for each object to store the input number. Note that the numbers we input can be showed or computed the lowest four digits due to the limited space in normal world. We give the normal world a space whose length is 4. So it just can show and calculate the numbers whose digits are less than 4. The part of larger than the fourth digits cannot be showed and calculated. The array, integer[digits], is used to store the input numbers and to be computed in the normal world. We overload the addition function so that it can be used to operate the addition in the normal world. If the result overflows when we use the addition function, it will produce an interruption. We use the int type flag1 to represent the status of the interruption. The initial value of flag1 is equal to 0. Once the interruption is produced, the value of flag1 will be changed to 1. When the SMC check the flag1 changes into 1, it will notify the monitor in the secure world and copy the data to the monitor. The normal world function will implement the computation in the normal world. If there exists a overflow, it will call the overflow function that tell the user the result overflow and there has a danger. The function ChangetoInt is also used to change the result into int type and assign the result to the inputvalue variable. The functions, setvalue and getvalue, are separately used to set the inputvalue variable and get the inputvalue variable. The operator<< is used to output the result which is computed in the normal world.

4.4 Process of Executing Program

The program will initiate the normal world, and receive the two input numbers. We will show the space given to the two numbers in the normal world. Then, the program will check if the digits of the two numbers are larger than 4.

(1) If they are smaller than 4, the program will start to execute the addition function for these two numbers.

   a. In the process of the addition, if the result overflows which means the number of digit is larger than 4, it will produce an interruption. And if the SMC instruction check there exist an instruction, it will copy the data to the monitor. After receiving the data, the monitor transfers the data to the secure world. The secure world gets the data, it will call the addition function to compute the data. When attaining the result, the secure world will send the result back to the normal world through monitor by using the SMC instruction.

   b. In the process of the addition, if the result doesn't overflow which means the number of digit isn't bigger than 4, it will stay in the normal world.

(2) If they are larger than 4, it will change the value of the flag1 which makes an interruption. When the SMC instruction checks the interruption, it will copy the data to the monitor. After receiving the data, the monitor transfers the data to the secure world. The secure world gets the data, it will call the addition function to compute the data. When attaining the result, the secure world will send the result back to the normal world through monitor by using the SMC instruction.

4.5 Workflow Chart

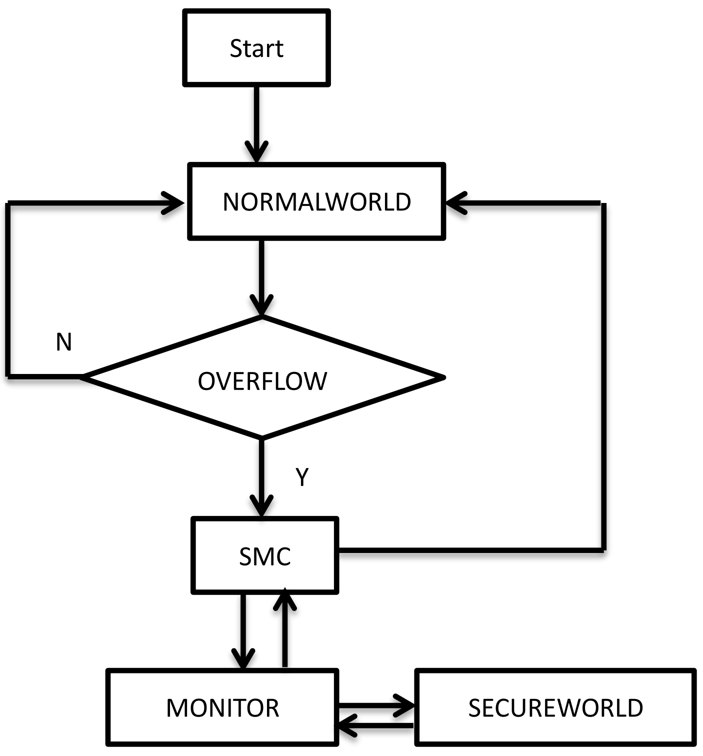


Figure 4.1

This is the flow chart of the program. It shows how the data transfer between the two worlds.

4.6 Core Code

Now we explain four core codes about our program, which are about the normalWorld, secureWorld, SMC and Monitor because these code implement the main function of the program. The explanations are in the following codes.

This is the core code about the normalWorld:

```
Trustzone Trustzone::normalWorld(Trustzone &op1, Trustzone &op2)
{
        cout << "\n*********************************" << endl;
        cout << "normal world open!!!" << endl;
        cout << "*********************************" << endl;
        int intnum=0;
        Trustzone op3;
        cout << "show the resource in the normal world" << endl;
        cout << "the resource for the first number:" << endl;
```

```
                    for (int i = 0; i < Trustzone::digits; i++)
                      //show the resource given to the first number in the normal
world
                    {
                        cout << op1.integer[i];
                    }
                    cout << "\nthe resource for the second number:" << endl;
                    for (int i = 0; i < Trustzone::digits; i++)
                      //show the resource given to the second number in the normal
world
                    {
                        cout << op2.integer[i];
                    }
                    if ((op1.getvalue()>=10000) || (op2.getvalue() >= 10000))
                      // if one of the number of digits of inputting figures is larger
than 4
                    {
                        flag1 = 1;//produce an interruption
                          CannotContain();//warning: the number is too big
                        SMC(op1,op2);//SMC instruction check the interruption
                        flag1 = 0;
                          //after getting result from the secure world, it cancels the
interruption
                        Trustzone op4(op1.getvalue());
                          //store the result from secure world in the normal world
                        return op4;
                    }
                    else
                    {
                        op3 = op1 + op2;// addition function
                        cout << "\nthe result of operation in the normal world" << endl;
                        cout << op3 << endl;
                        if (flag1 == 1)// if there exists an interruption
                        {
                             Trustzone op4(op1.getvalue());
                               //store the result from secure world in the normal
world
                             flag1 = 0;// reset the flag1 and cancel the interruption
                             return op4;
                        }
                        else
                        {
                             op3.setvalue(ChangetoInt(op3));//change the result to int
type
```

```
                              Trustzone op4(op3.getvalue());
                                //store the result from secure world in the normal
world
                              return op4;
                         }

                    }
}
```

This is the core code about the secure world:

```
int TrustzonesecureWorld::secureWorld(Trustzone &op1, Trustzone &op2)
{
                int change = 0;
                 TrustzonesecureWorld temp(0);
                cout << "\n**********************************" << endl;
                cout << "secure world open!!!" << endl;
                cout << "**********************************" << endl;
                TrustzonesecureWorld n1(op1);
                TrustzonesecureWorld n2(op2);
                cout << "show the resource in the secure world" << endl;
                cout << "the resource for the first number:" << endl;
        for (int i = 0; i < TrustzonesecureWorld::maxdigits; i++)
        //show the resource given to the first number in the secure world
        {
                cout << n1.inputvalue[i];
        }
        cout << "\nthe resource for the second number:" << endl;
        for (int i = 0; i < TrustzonesecureWorld::maxdigits; i++)
        //show the resource given to the second number in the secure world
        {
                cout << n2.inputvalue[i];
        }
        cout << "\nthe resource for the result of calculation in the secure world:" <<
endl;
        for (int i = 0; i < TrustzonesecureWorld::maxdigits; i++)
        //show the resource given to the result (the total of first and second
number)
        {
                cout << (n1+n2).inputvalue[i];
        }
        cout << "\nthe result of operation in the secure world:" << endl;
        cout << "\n" << n1 + n2 << endl;
        temp = n1 + n2;
        change=ChangetoInt(temp);//change the result into int type
        return change;//return the result to the normal world through monitor
}
```

This is the core code about the monitor:

```
int TrustzonesecureWorld::Monitor(Trustzone & op1, Trustzone & op2)
{
            int temp;//store the result from the secure world
            cout << "\nmonitor mode open due to existing interruption" << endl;
            temp=secureWorld(op1, op2); //transfer the data to the secure world
and get the result from the secure world
            return temp;
}
```

This is the core code about the SMC:

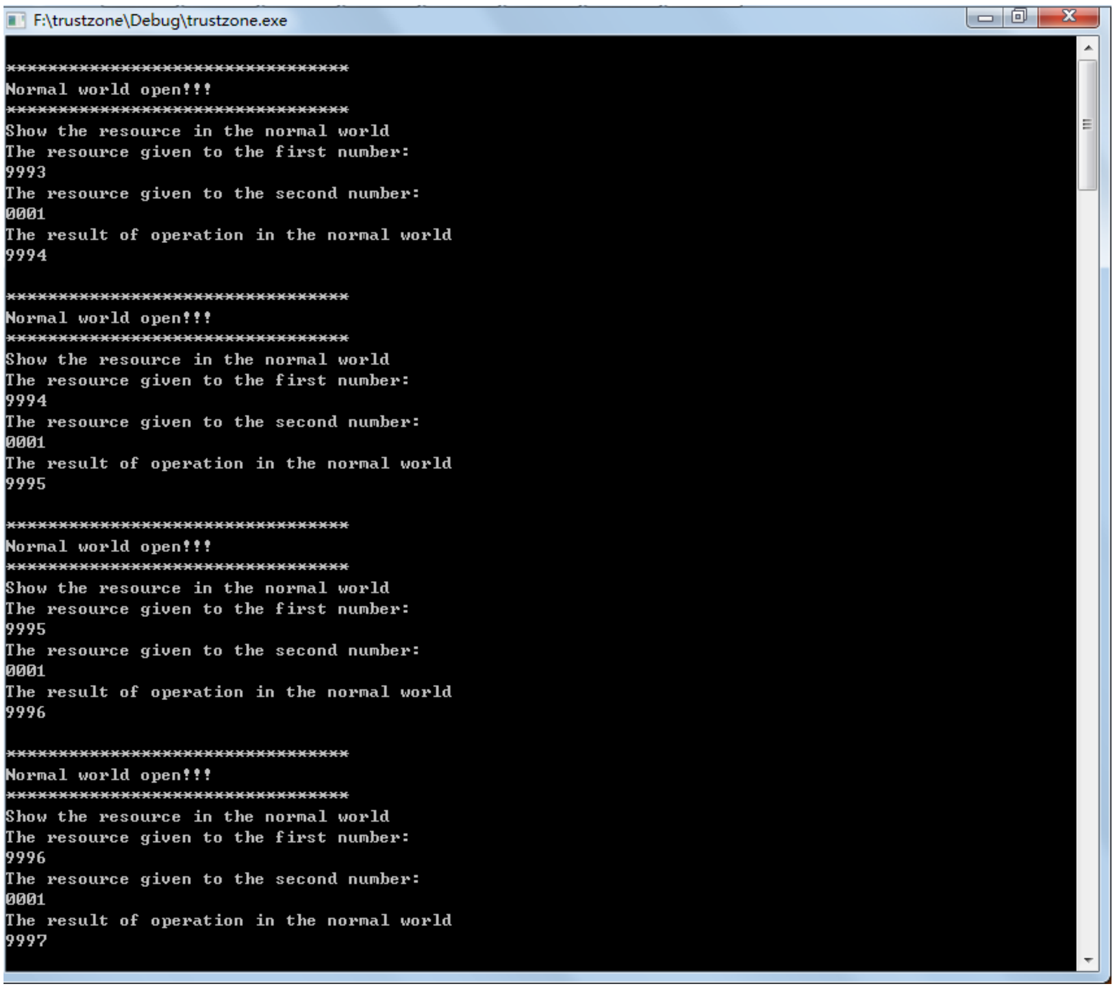
```
int Trustzone::SMC(Trustzone & op1, Trustzone & op2)
{
            int temp;//use store the computation result from the secure world
            if (flag1==1)//if flag1==1, it means there exists an interruption
            {
               cout << "\ntransferring to the monitor mode due to interruption" <<
endl;
               TrustzonesecureWorld SMC_handler;//build a SMC handler to
transfer the data
               temp=SMC_handler.Monitor(op1, op2);//get the result from secure
world
               op1.setvalue(temp); //store the result in variable, inputvalue
                 cout << "\n*********************************" << endl;
               cout << "welcome back to normal world" << endl;
               cout << "*********************************" << endl;
            }
            return 0;
}
```

4.7 Test

Now we begin to execute the program and test the result. After getting the result, we use the screenshots to explain the test.

```
Trustzone trustzone;
Trustzone trustzone1(9993);
Trustzone trustzone2(1);
Trustzone trustzone3(99999);
Trustzone trustzone4(99999);
for (int i = 0; i < 10; i++)
{
  trustzone1=trustzone.normalWorld(trustzone1, trustzone2);
}
trustzone.normalWorld(trustzone3, trustzone4);
system("pause");
```

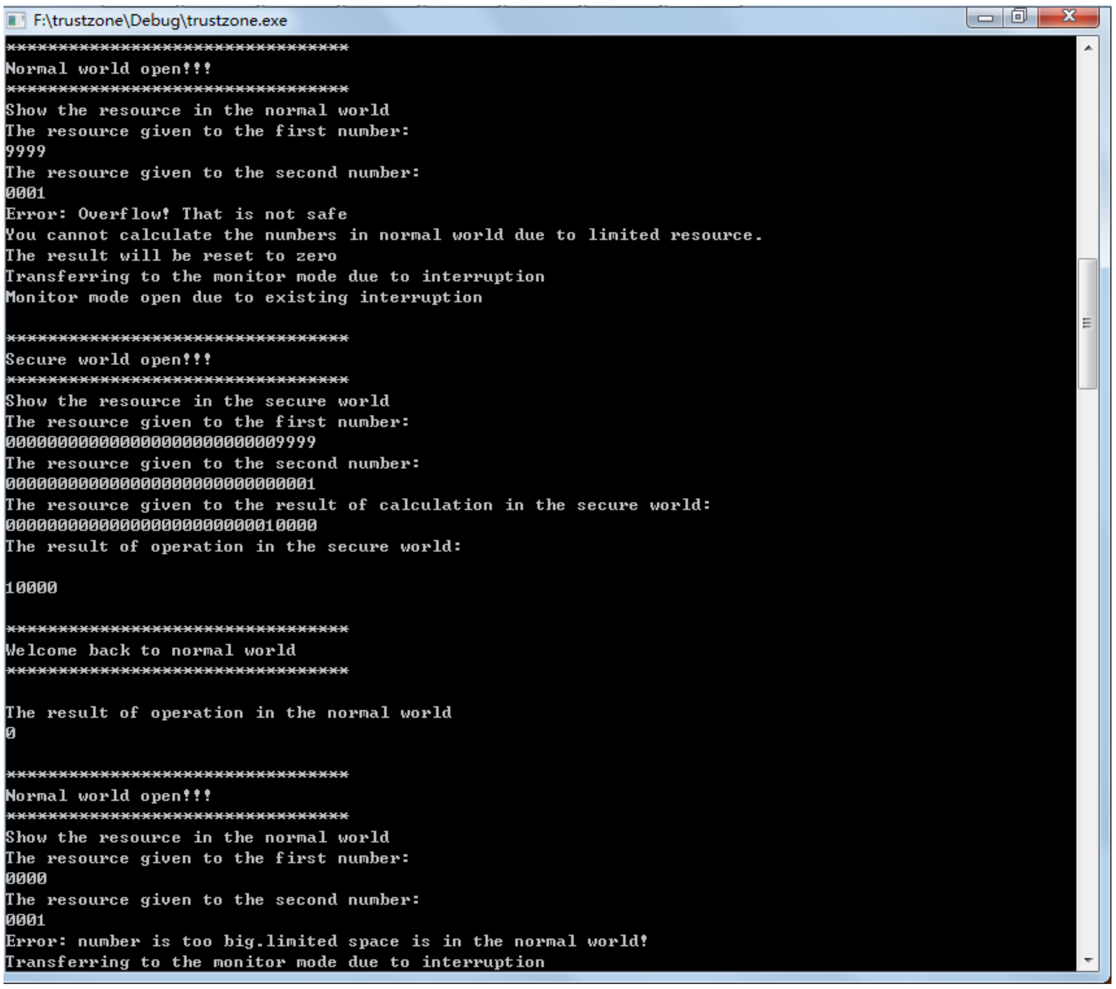
In the main function, we give the four value to the normal world. According to the mechanism of the Trustzone, we may get the result:
9993+1=9994; 9994+1=9995; 9995+1=9996; 9996+1=9997; 9997+1=9998; 9998+1=9999; 9999+1=10000; 10000+1=10001; 10001+1=10002; 10002+1=10003. And 99999+99999=199998.
Now, we get the screen shot:



```
F:\trustzone\Debug\trustzone.exe

***********************************
Normal world open!!!
***********************************
Show the resource in the normal world
The resource given to the first number:
9993
The resource given to the second number:
0001
The result of operation in the normal world
9994

***********************************
Normal world open!!!
***********************************
Show the resource in the normal world
The resource given to the first number:
9994
The resource given to the second number:
0001
The result of operation in the normal world
9995

***********************************
Normal world open!!!
***********************************
Show the resource in the normal world
The resource given to the first number:
9995
The resource given to the second number:
0001
The result of operation in the normal world
9996

***********************************
Normal world open!!!
***********************************
Show the resource in the normal world
The resource given to the first number:
9996
The resource given to the second number:
0001
The result of operation in the normal world
9997
```

Figure 4.2

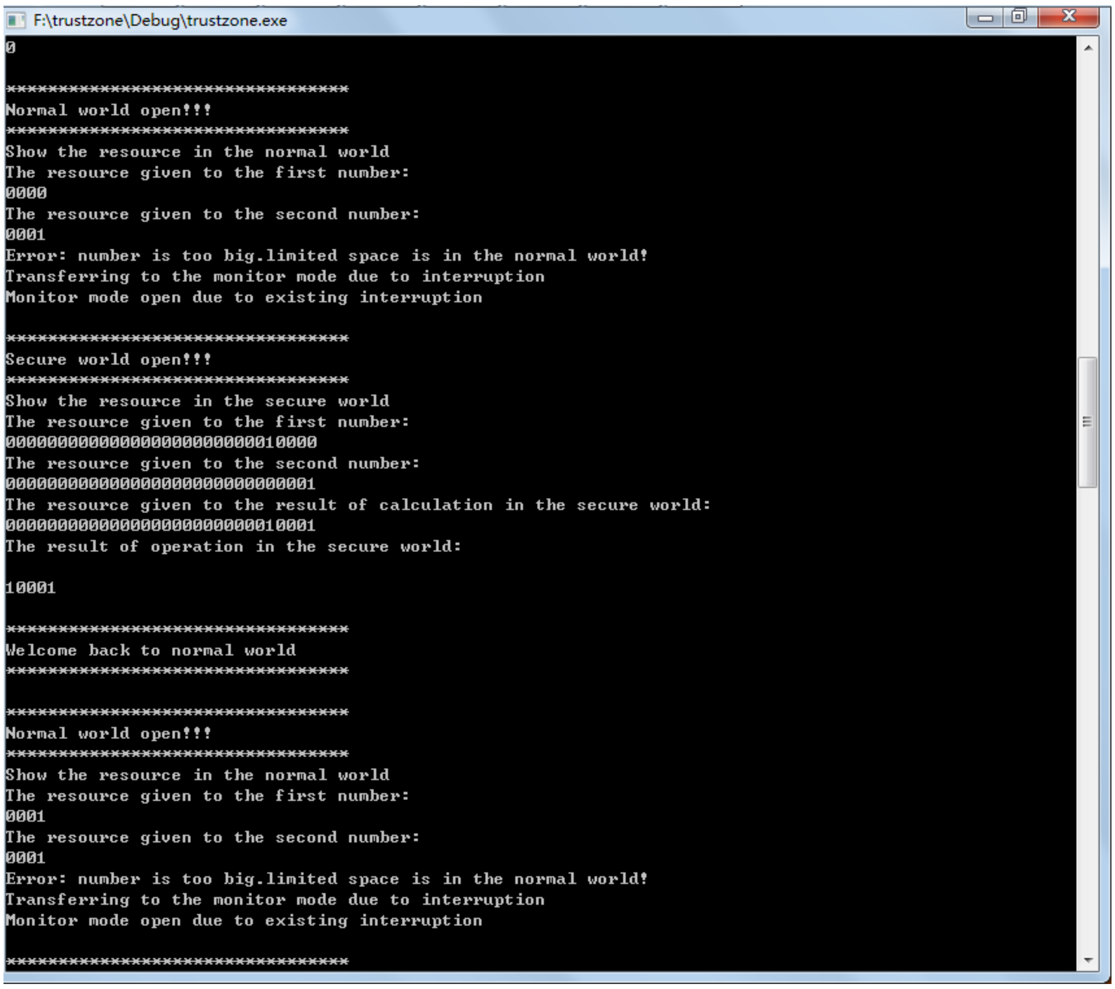
From the Figure 4.2, you can see the program show how many space it gives to each object. It only provides the lowest 4 digits to each object. And the program begins in the normal world. we can know the operation stay in the normal world all the time when it calculates the 9993+1=9994; 9994+1=9995; 9995+1=9996; 9996+1=9997; 9997+1=9998; 9998+1=9999. This is mainly because there is enough resource to compute these numbers safely in the normal world. There is no need to transfer to the secure world and use its resource to computation.



```
***********************************
Normal world open!!!
***********************************
Show the resource in the normal world
The resource given to the first number:
9999
The resource given to the second number:
0001
Error: Overflow! That is not safe
You cannot calculate the numbers in normal world due to limited resource.
The result will be reset to zero
Transferring to the monitor mode due to interruption
Monitor mode open due to existing interruption

***********************************
Secure world open!!!
***********************************
Show the resource in the secure world
The resource given to the first number:
000000000000000000000000009999
The resource given to the second number:
000000000000000000000000000001
The resource given to the result of calculation in the secure world:
000000000000000000000000010000
The result of operation in the secure world:

10000

***********************************
Welcome back to normal world
***********************************

The result of operation in the normal world
0

***********************************
Normal world open!!!
***********************************
Show the resource in the normal world
The resource given to the first number:
0000
The resource given to the second number:
0001
Error: number is too big.limited space is in the normal world!
Transferring to the monitor mode due to interruption
```

Figure 4.3

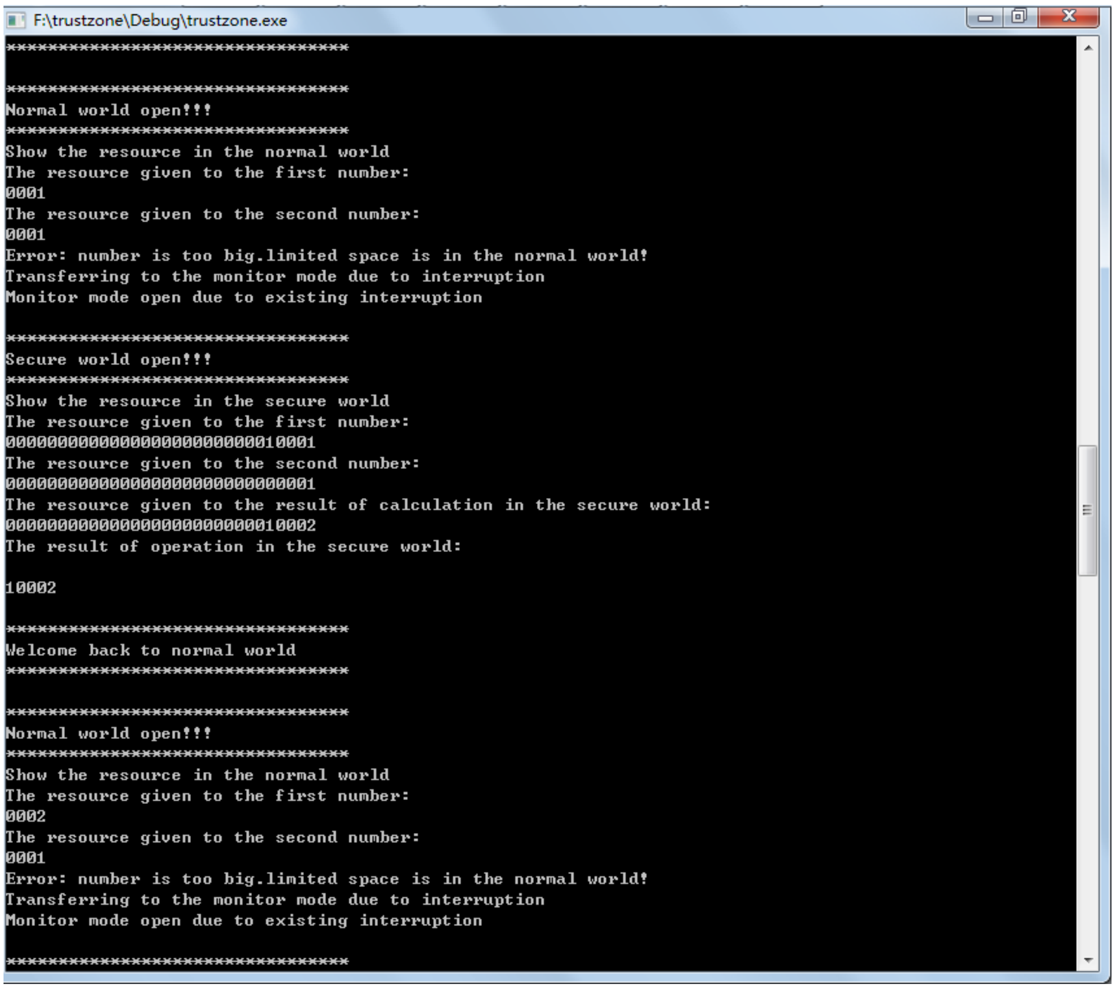
From the figure 4.3, when computing 9999+1, the program produces an error, " Error: Overflow! That is not safe. You cannot calculate the numbers in normal world due to limited resource. The result will be reset to zero", because there exists overflow due to limited space. And it also produce an interruption which can be checked by SMC instruction and SMC instruction copies the data and transfers the data to the monitor mode in the secure world. So you can see the sentence, " Transferring to the monitor mode due to interruption". When it enters the secure world from normal world, you can see this sentence, "Monitor mode open due to existing interruption". It means that the data have entered the secure world. Thus, the screen show secure world open and the space it gives to the data. The length of space is 30. And the computation will be executed by using the enough space in the secure world. You can see the program shows the 5 digits result, 10000, in the secure world. After getting the calculation result in the secure world, it will go back to the normal world through monitor. So you can see the sentence, "Welcome back to normal world". The result computed in the normal world will be reset to 0 due to no enough resource for computation. Now, in the normal world, the program continues to computing which means it calculate 10000+1. But, with limited resource, we just can see the lowest 4 digits of 10000, which is showed 0000. So it means we cannot compute the numbers in the normal world because we will get the wrong result. It makes an interruption and error like what I explained before, and it will transfer to the secure world by monitor and program uses the resource of secure world to compute and return to the normal world through monitor.



```
F:\trustzone\Debug\trustzone.exe
0

*********************************
Normal world open!!!
*********************************
Show the resource in the normal world
The resource given to the first number:
0000
The resource given to the second number:
0001
Error: number is too big.limited space is in the normal world!
Transferring to the monitor mode due to interruption
Monitor mode open due to existing interruption

*********************************
Secure world open!!!
*********************************
Show the resource in the secure world
The resource given to the first number:
000000000000000000000000010000
The resource given to the second number:
000000000000000000000000000001
The resource given to the result of calculation in the secure world:
000000000000000000000000010001
The result of operation in the secure world:

10001

*********************************
Welcome back to normal world
*********************************

*********************************
Normal world open!!!
*********************************
Show the resource in the normal world
The resource given to the first number:
0001
The resource given to the second number:
0001
Error: number is too big.limited space is in the normal world!
Transferring to the monitor mode due to interruption
Monitor mode open due to existing interruption

*********************************
```

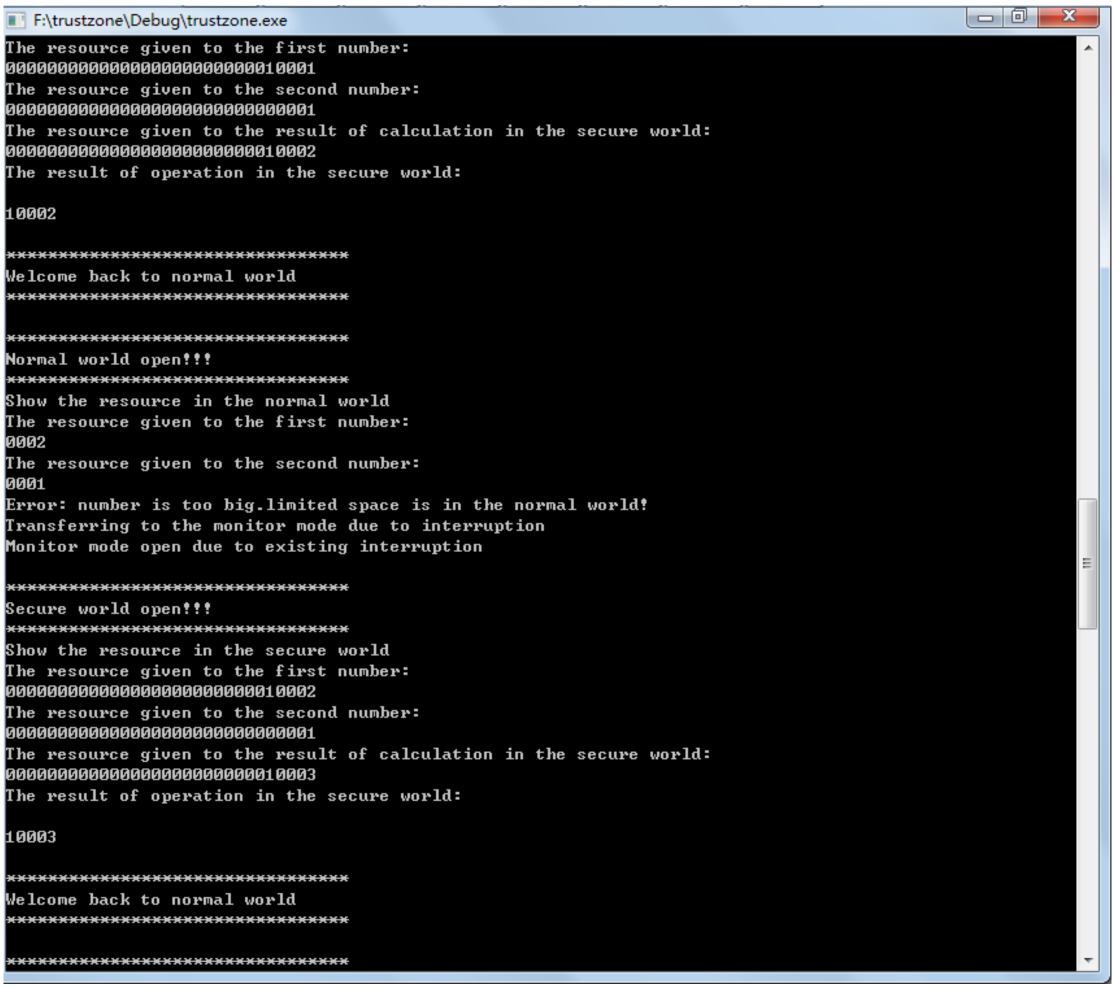
Figure 4.4

Figure 4.4 shows the result of 10000+1. And if we continue the computation, 10001+1, 10002+1, the following processes will be the same as 10000+1. Now we use the “for” loop to implement the function. The results will be showed in the figure 4.5 and figure 4.6.


F:\trustzone\Debug\trustzone.exe

```
*********************************

*********************************
Normal world open!!!
*********************************
Show the resource in the normal world
The resource given to the first number:
0001
The resource given to the second number:
0001
Error: number is too big.limited space is in the normal world!
Transferring to the monitor mode due to interruption
Monitor mode open due to existing interruption

*********************************
Secure world open!!!
*********************************
Show the resource in the secure world
The resource given to the first number:
000000000000000000000000010001
The resource given to the second number:
000000000000000000000000000001
The resource given to the result of calculation in the secure world:
000000000000000000000000010002
The result of operation in the secure world:

10002

*********************************
Welcome back to normal world
*********************************

*********************************
Normal world open!!!
*********************************
Show the resource in the normal world
The resource given to the first number:
0002
The resource given to the second number:
0001
Error: number is too big.limited space is in the normal world!
Transferring to the monitor mode due to interruption
Monitor mode open due to existing interruption

*********************************
```

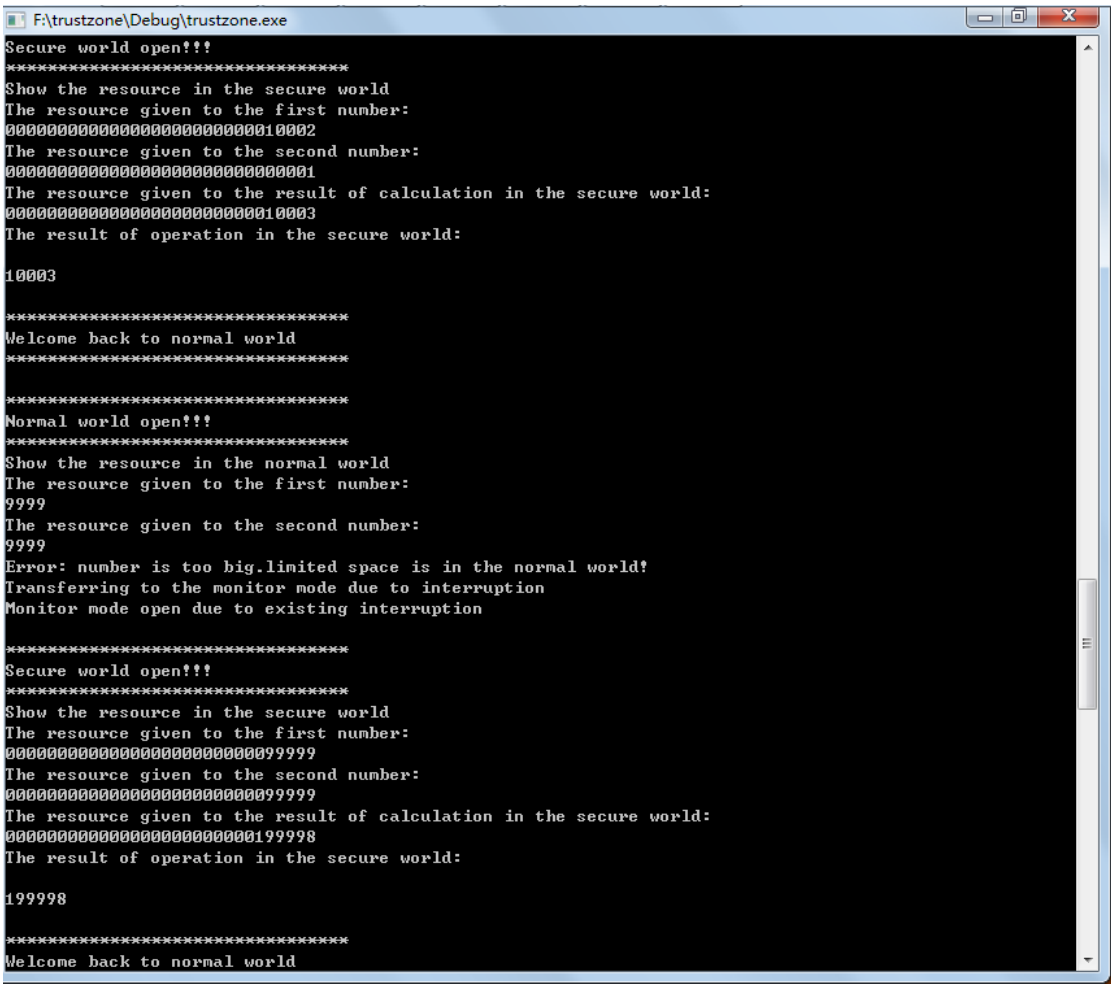
Figure 4.5



```
F:\trustzone\Debug\trustzone.exe
The resource given to the first number:
000000000000000000000000010001
The resource given to the second number:
000000000000000000000000000001
The resource given to the result of calculation in the secure world:
000000000000000000000000010002
The result of operation in the secure world:

10002

*********************************
Welcome back to normal world
*********************************

*********************************
Normal world open!!!
*********************************
Show the resource in the normal world
The resource given to the first number:
0002
The resource given to the second number:
0001
Error: number is too big.limited space is in the normal world!
Transferring to the monitor mode due to interruption
Monitor mode open due to existing interruption

*********************************
Secure world open!!!
*********************************
Show the resource in the secure world
The resource given to the first number:
000000000000000000000000010002
The resource given to the second number:
000000000000000000000000000001
The resource given to the result of calculation in the secure world:
000000000000000000000000010003
The result of operation in the secure world:

10003

*********************************
Welcome back to normal world
*********************************

*********************************
```

Figure 4.6

Now we continue using the numbers, 99999 and 99999, to calculate. The result will be shown below in the figure 4.7. You can see the 99999 and 99999 in the normal world will be stored to be 9999 and 9999 due to limited space for them. If we the program uses the numbers to compute in the normal world, it must get the wrong answer. With the mechanism of Trustzone, if 99999 and 99999 in the normal want to take part in the computation, it will produce a signal of interruption. The signal will use the SMC instruction to activate the monitor mode in secure world. And the data will be copied into the secure world through monitor by using SMC instruction. And we can find in the secure world the numbers, 99999 and 99999, participating in the computation are shown their original values. When we execute the computation, we can get the right result. Finally, it will come back the normal world again. The result shows the sentence in screen, “Welcome back to normal world”.



```
Secure world open!!!
*********************************
Show the resource in the secure world
The resource given to the first number:
000000000000000000000000010002
The resource given to the second number:
000000000000000000000000000001
The resource given to the result of calculation in the secure world:
000000000000000000000000010003
The result of operation in the secure world:

10003

*********************************
Welcome back to normal world
*********************************

*********************************
Normal world open!!!
*********************************
Show the resource in the normal world
The resource given to the first number:
9999
The resource given to the second number:
9999
Error: number is too big.limited space is in the normal world!
Transferring to the monitor mode due to interruption
Monitor mode open due to existing interruption

*********************************
Secure world open!!!
*********************************
Show the resource in the secure world
The resource given to the first number:
000000000000000000000000099999
The resource given to the second number:
000000000000000000000000099999
The resource given to the result of calculation in the secure world:
000000000000000000000000199998
The result of operation in the secure world:

199998

*********************************
Welcome back to normal world
```

Figure 4.7

4.8 Conclusion

The program uses the example of computation to stimulate the working mechanism of the Trustzone. Without the mechanism, it will get the wrong answer in the computation because in the normal world the system just gives the numbers limited resource. With the help of the mechanism, we can calculate big data or small data and get the right answer by switching between normal world and secure world that has enough resource. So the mechanism keeps the computation on the right way.

5.One-Time Password Use Case

In this project, we want to design a demo of One-time Password system to take advantage of TrustZone features.

5.1 Context

A commonly used form of one-time passwords is *Transaction Authentication Numbers* (TANs) [8] for online banking purposes. Today, some banks send paper cards to their customers with a list of TANs. When a customer initiates an online transfer, the bank specifies an index into the TAN list and asks for the associated TAN. The customer, in addition to typing their personal password, must respond with the correct TAN, otherwise the transaction is aborted. Each TAN is used once; when all TANs are used, the bank sends a new TAN list to the customer.

In One-Time Password system, a paper cards to their customers with a list of TANs could be replaced by an electronic list that given by the bank. That is

- Banks can send a TAN list to customers' own smartphone which be authorize by customers.
- The TAN list will be stored in kernel of smartphones, which can be treated as a secure world of smartphone
- When a customer initiates an online transfer, the bank specifies an index to customer's smartphone and asks for the associated TAN. After customer input password, the bank's app in the smartphone will access the TAN list and get the associated TAN.
- Because, the TAN list is unique for every customer, so if other people have a customer's password but don't have that customer's phone, they still cannot do any transaction of that account.
- Every TAN in the list can only be used once. After using, the TAN in list in Bank Sever will be deleted, as well as the one in the TAN list stored in Secure World of customers mobile.

5.2 Design

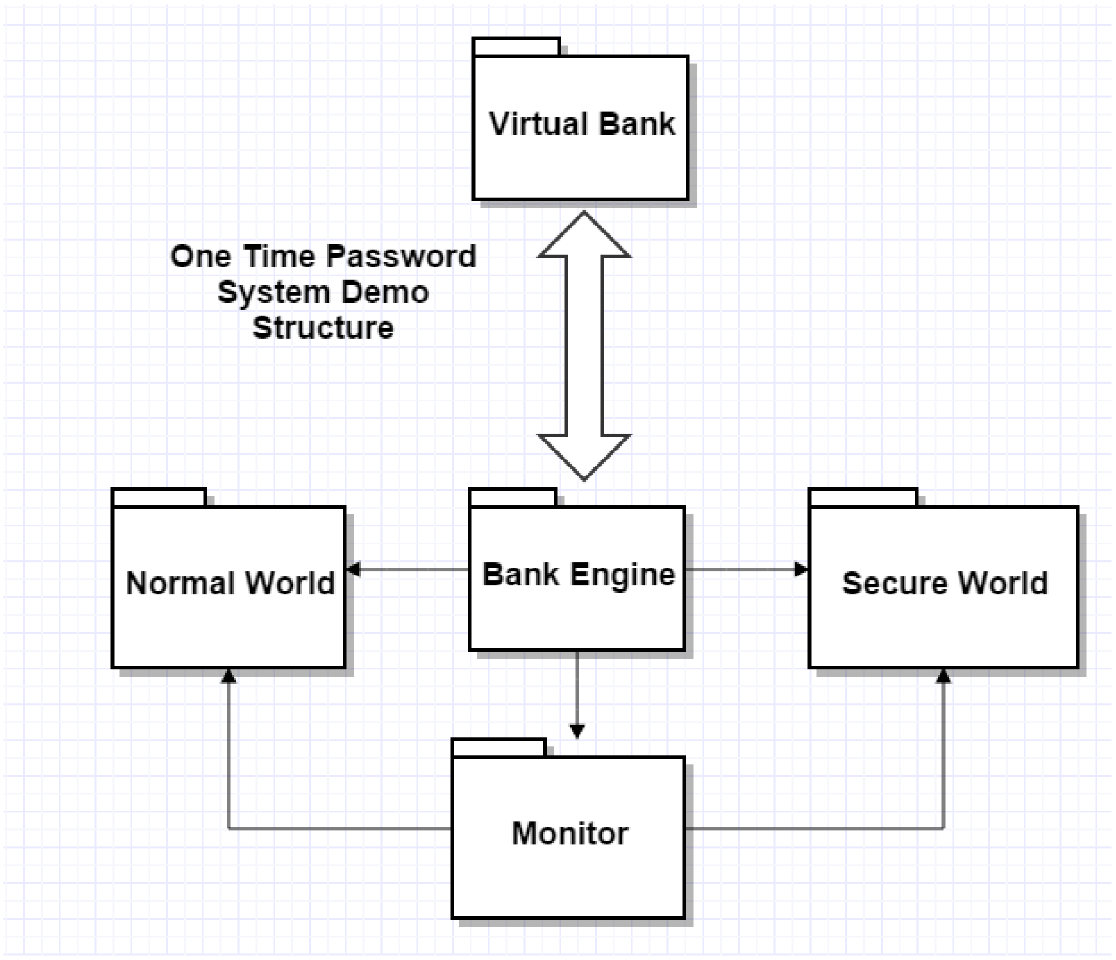
1. **Partition off a small part of the application that needs to handle sensitive data into a secure world.**
   A copy of Transaction Authentication Numbers (TAN) list which given by the bank should be the most important sensitive data, because user not only need his own password for login, but also a TAN and these TAN list is unique to every customers. It should be stored in a save place. In this case, decide that TAN list and the function to handle TAN list should be into a secure world.
2. **Deploy the secure world on the designated platform**
   Because even customers themselves normally do not have the right to access the kernel of devices, so it's fine to deploy the secure world on the kernel of smartphones. (Customers do not need to know what the TAN list, access the TAN list or know what the numbers are, they just keep the TAN list).
3. **Ensure that sensitive data can only be accessed on a designated platform by sealing the data to the secure world running on the designated platform**

This system should provide three parts: Normal World, Secure World, and Monitor. Users can only access normal world of the application. When they need the TAN list, normal world will send a request to monitor, then monitor to decide whether or not to change mode to secure world to do the work.

4. **Deploy the sealed data to the designated platform to run it inside its secure world, thereby ensuring that the secure world is protected at runtime.**
   The TAN list in the secure world can only accept the data from Bank server, it can only be read by monitor, cannot be modified in any time.

5.3 Demo Structure

The demo of this One Time Password System including five parts:



Virtual Bank Package:
Simulates the Bank Server, which creates TAN list for customers, provides index to customers, checks the combo of customers' password and TAN, and deletes the TAN that has been used.

Bank Engine Package:
An interface for customers. It provides communication service to Virtual Bank, get TAN list from Virtual Bank and store it to Secure World, get password and TAN from

Normal World, dispatch functions to Monitor and Normal World.

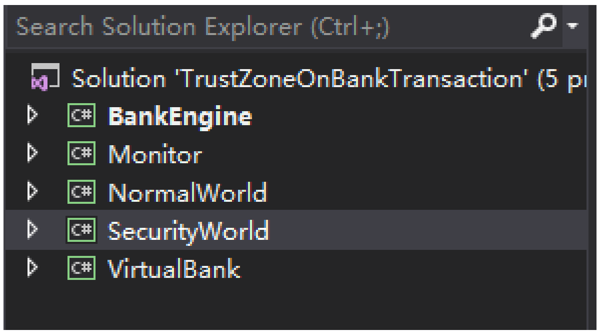
Secure World:
The package that stores the TAN list and does the calculation for sensitive data. It cannot be accessed by Normal World directly.

Normal World:
Receive index from Virtual Bank and send requests to monitor for mode changing, and get the TAN back for succeed login and traction.

Monitor:
Monitor the data flow and mode change between two worlds. Deploy mode changes

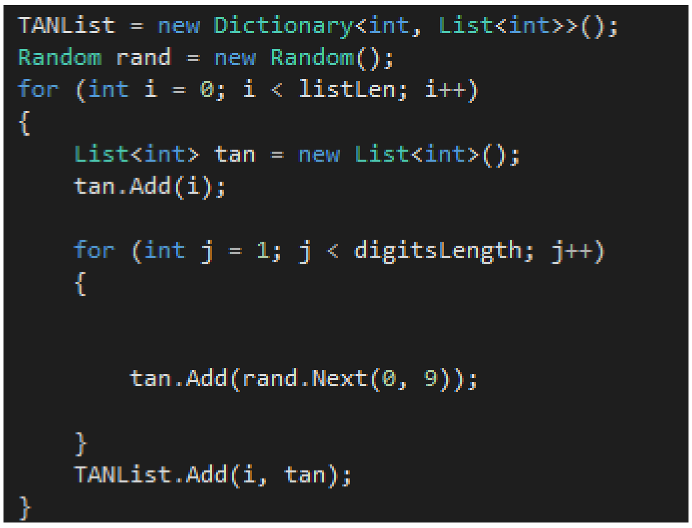
5.4 Deployment of the Demo in C#

- For each packages, we design a project in C# for it. All of them combine in a solution in C# that simulates the One Time Password System. The figure shows as below.



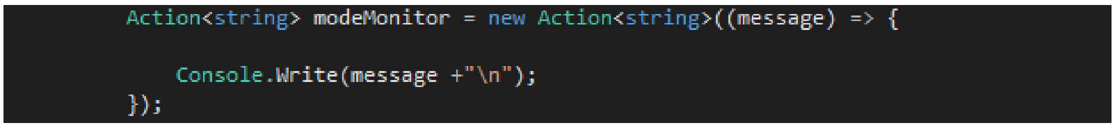
Search Solution Explorer (Ctrl+;)
Solution 'TrustZoneOnBankTransaction' (5 p
BankEngine
Monitor
NormalWorld
SecurityWorld
VirtualBank

- First Virtual Bank Server would build a TAN list, store it and send a copy to an authorized phone’s Secure World. Secure World would also save this TAN list in kernel memory. Below is the code for create such TAN with 20digits random number.



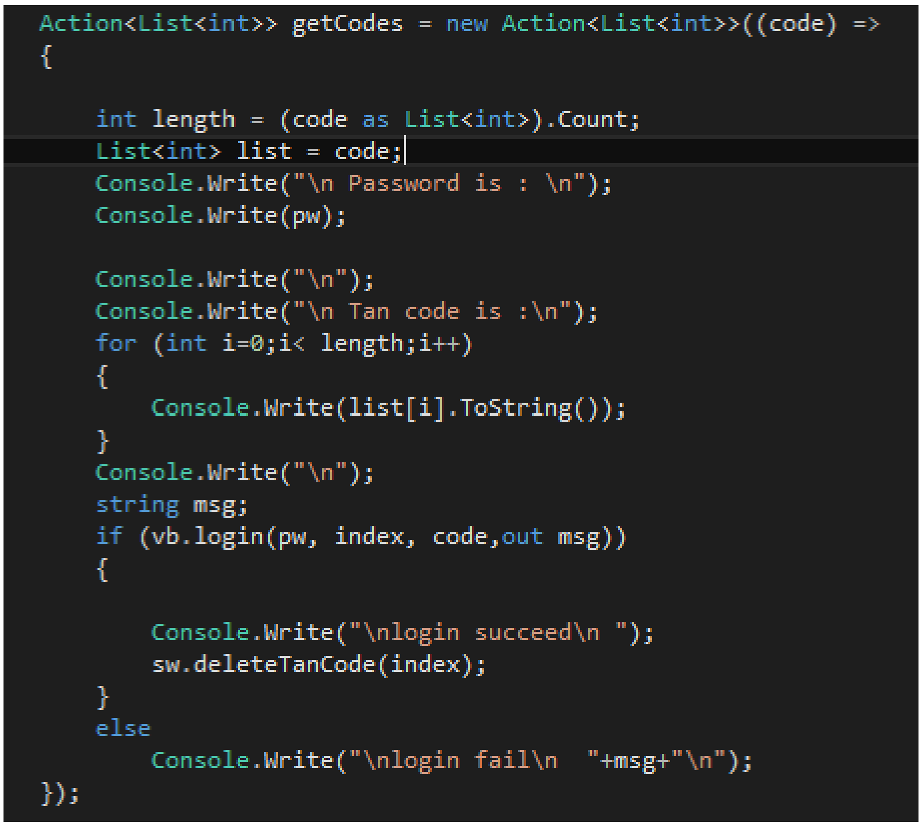
```
TANList = new Dictionary<int, List<int>>();
Random rand = new Random();
for (int i = 0; i < listLen; i++)
{
    List<int> tan = new List<int>();
    tan.Add(i);

    for (int j = 1; j < digitsLength; j++)
    {


        tan.Add(rand.Next(0, 9));

    }
    TANList.Add(i, tan);
}
```

- In the initialization of Bank Engine, it sets up call back function for normal world and monitor.



```
        Action<string> modeMonitor = new Action<string>((message) => {

            Console.Write(message +"\n");
        });
```

The call back function above can return message from monitor to Bank Engine interface.

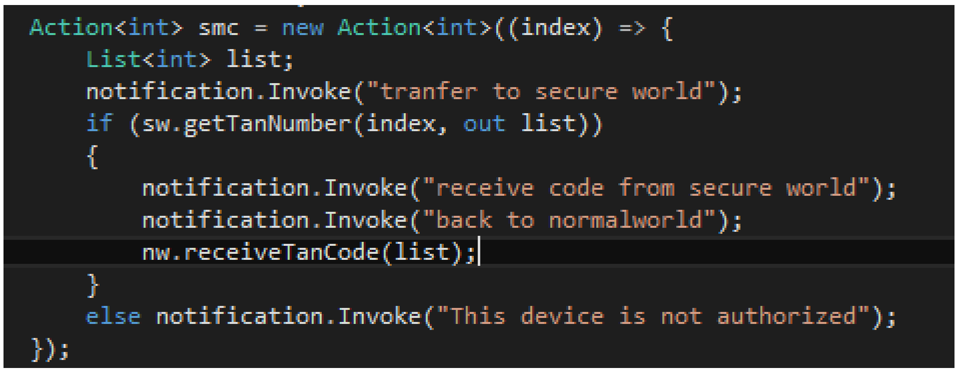


```
Action<List<int>> getCodes = new Action<List<int>>((code) =>
{

    int length = (code as List<int>).Count;
    List<int> list = code;
    Console.Write("\n Password is : \n");
    Console.Write(pw);

    Console.Write("\n");
    Console.Write("\n Tan code is :\n");
    for (int i=0;i< length;i++)
    {
        Console.Write(list[i].ToString());
    }
    Console.Write("\n");
    string msg;
    if (vb.login(pw, index, code,out msg))
    {

        Console.Write("\nlogin succeed\n ");
        sw.deleteTanCode(index);
    }
    else
        Console.Write("\nlogin fail\n  "+msg+"\n");
});
```

The call back function above is for Normal World to return the TAN to Bank Engine. Once the TAN has been sent to Bank Engine, the Bank Engine will send the user password along with index given by Bank Server and TAN to Bank

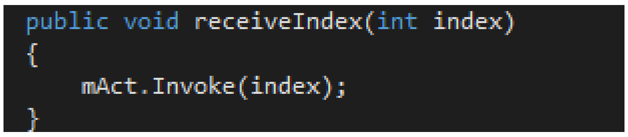
Server, then can the result back. An advantage for call back function is that, the call back functions determine what Monitor and Bank need and Normal World can only use these function for so called communication, but no other function can be customized by Normal World.

- When Normal World gets the index from Virtual Bank Server, it would invoke monitor mode, then monitor mode change to Secure World to get the TAN in TAN list by using the index. The follow codes provide this service.
  Monitor.cs



```
Action<int> smc = new Action<int>((index) => {
    List<int> list;
    notification.Invoke("tranfer to secure world");
    if (sw.getTanNumber(index, out list))
    {
        notification.Invoke("receive code from secure world");
        notification.Invoke("back to normalworld");
        nw.receiveTanCode(list);
    }
    else notification.Invoke("This device is not authorized");
});
```

Normal World.cs



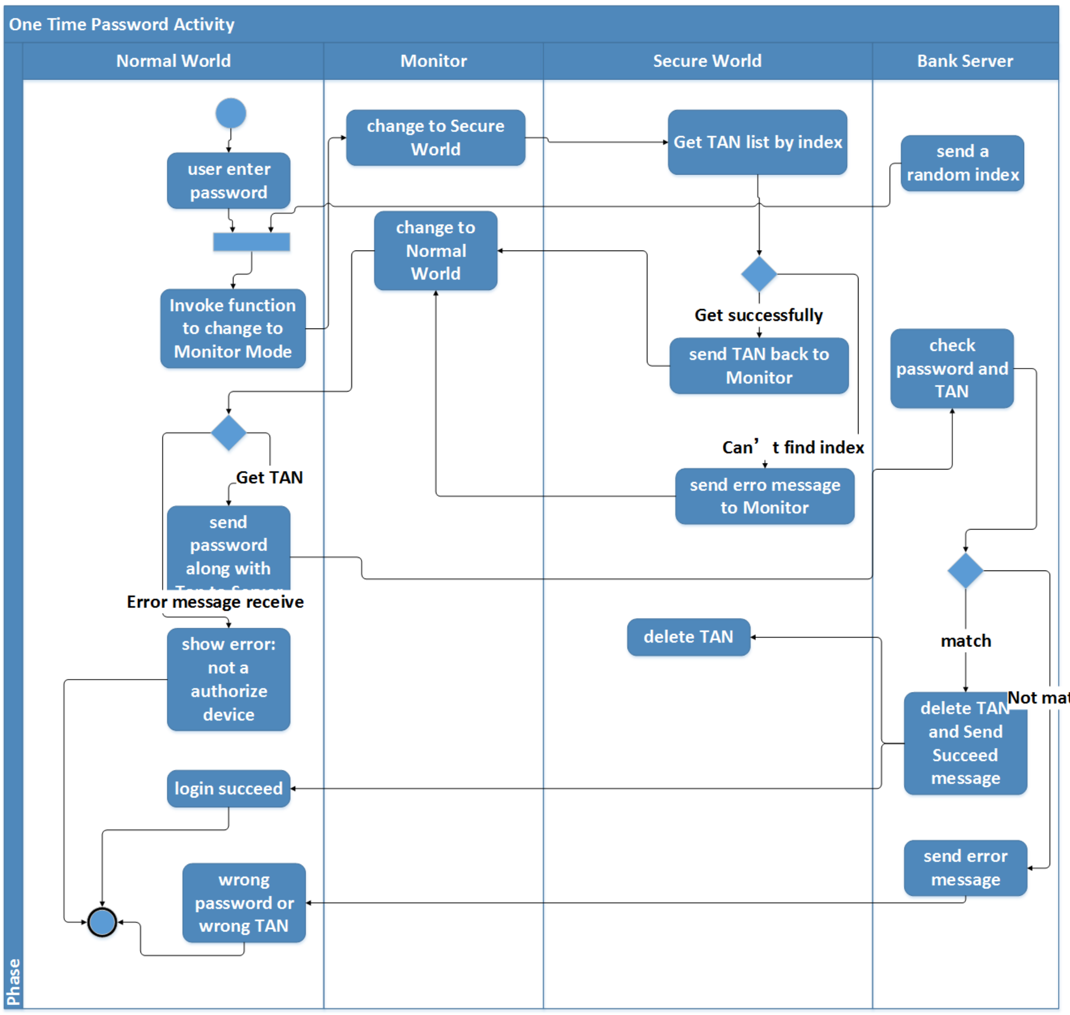
```
public void receiveIndex(int index)
{
    mAct.Invoke(index);
}
```

5.5 Activity:

The processes of a user check are as below:

- User input password
- Server send index to User
- Normal World use index to ask Monitor
- Monitor send index to Secure Word
- Secure Word query the TAN correspond to this index
  - If not find (not in a authorized device)
- Send TAN back to Normal World by Monitor
- Normal World send the combo of password and TAN to Server
- Server check the password and TAN
  - If password not match (wrong password)
  - If TAN not match (this device is not trusted)
- If match, delete the TAN in TAN list
- Delete the TAN in TAN list which is stored in Secure World
- Show success message

The details of this activity are as follow.



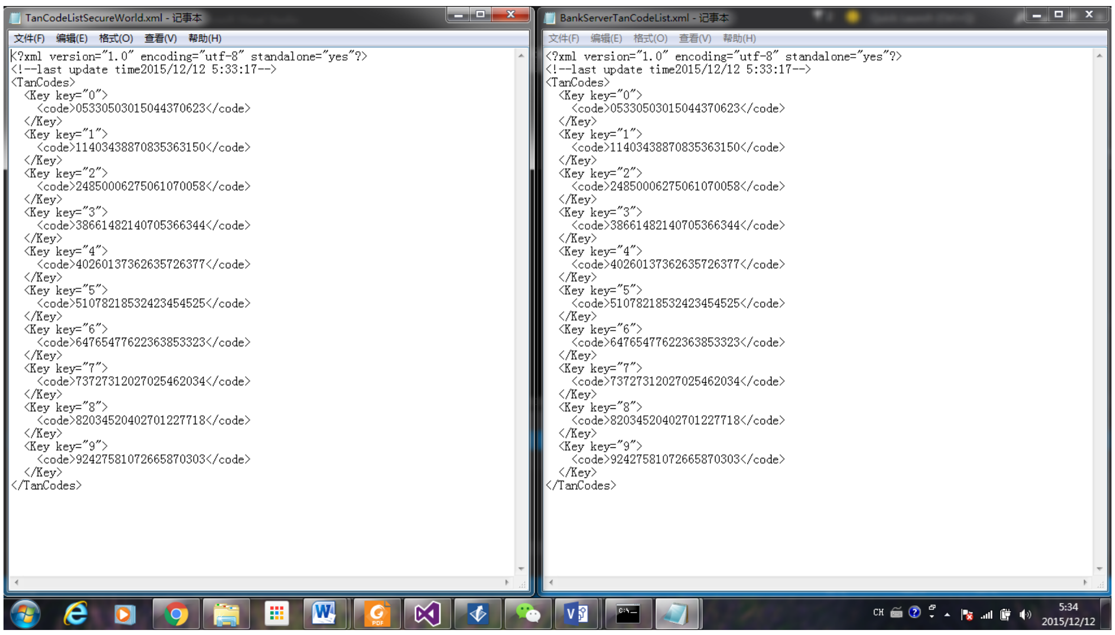
5.6 Results

- At the very first beginning, neither Bank Server nor Secure World in customer’s phone has TAN list. So Bank Server should create it first and send a copy to customer’s authorized phone. After running the project, two XML files will be creates in Bank Server and Users Secure World.

  BankServerTanCodeList.xml

  TanCodeListSecureWorld.xml

  Both of them have the same TAN list

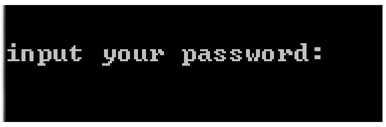

TanCodeListSecureWorld.xml - 记事本

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!--last update time2015/12/12 5:33:17-->
<TanCodes>
  <Key key="0">
    <code>05330503015044370623</code>
  </Key>
  <Key key="1">
    <code>11403438870835363150</code>
  </Key>
  <Key key="2">
    <code>24850006275061070058</code>
  </Key>
  <Key key="3">
    <code>38661482140705366344</code>
  </Key>
  <Key key="4">
    <code>40260137362635726377</code>
  </Key>
  <Key key="5">
    <code>51078218532423454525</code>
  </Key>
  <Key key="6">
    <code>64765477622363853323</code>
  </Key>
  <Key key="7">
    <code>73727312027025462034</code>
  </Key>
  <Key key="8">
    <code>82034520402701227718</code>
  </Key>
  <Key key="9">
    <code>92427581072665870303</code>
  </Key>
</TanCodes>
```

BankServerTanCodeList.xml - 记事本

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!--last update time2015/12/12 5:33:17-->
<TanCodes>
  <Key key="0">
    <code>05330503015044370623</code>
  </Key>
  <Key key="1">
    <code>11403438870835363150</code>
  </Key>
  <Key key="2">
    <code>24850006275061070058</code>
  </Key>
  <Key key="3">
    <code>38661482140705366344</code>
  </Key>
  <Key key="4">
    <code>40260137362635726377</code>
  </Key>
  <Key key="5">
    <code>51078218532423454525</code>
  </Key>
  <Key key="6">
    <code>64765477622363853323</code>
  </Key>
  <Key key="7">
    <code>73727312027025462034</code>
  </Key>
  <Key key="8">
    <code>82034520402701227718</code>
  </Key>
  <Key key="9">
    <code>92427581072665870303</code>
  </Key>
</TanCodes>
```

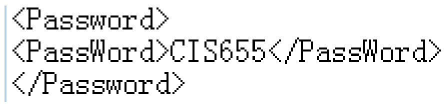
- After this, a console will show and ask user to input their password.



```
input your password:
```

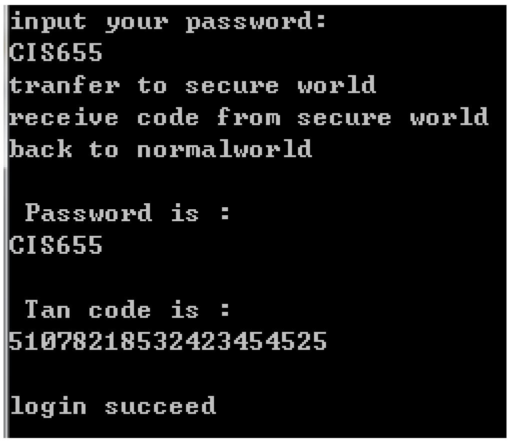
Password is configured by customers when they on a bank account. It's stored in Bank Server in an XML file

Let's set password is CIS655



```
<Password>
<PassWord>CIS655</PassWord>
</Password>
```

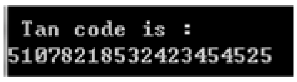
- With the correct copy of TAN list and a correct password, users can login successfully



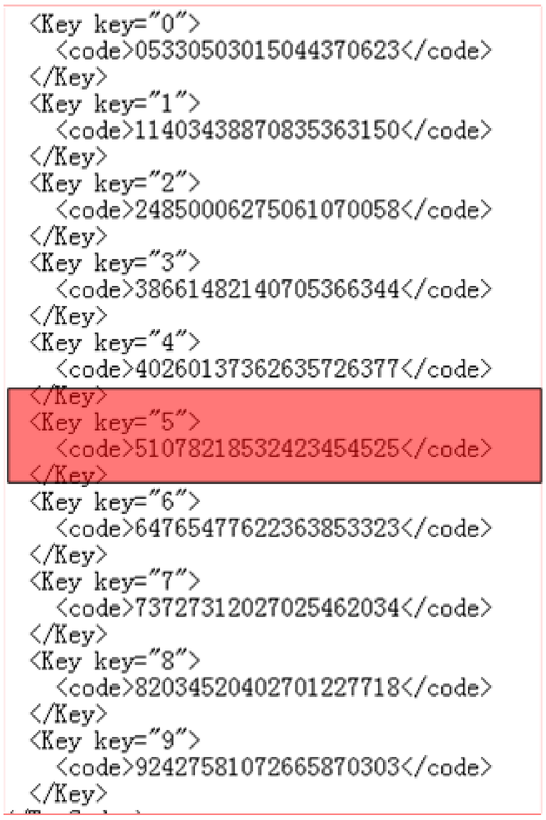
```
input your password:
CIS655
tranfer to secure world
receive code from secure world
back to normalworld

 Password is :
CIS655

 Tan code is :
51078218532423454525

login succeed
```

After successfully login, the TAN used to login will be delete from both the TAN list in Bank

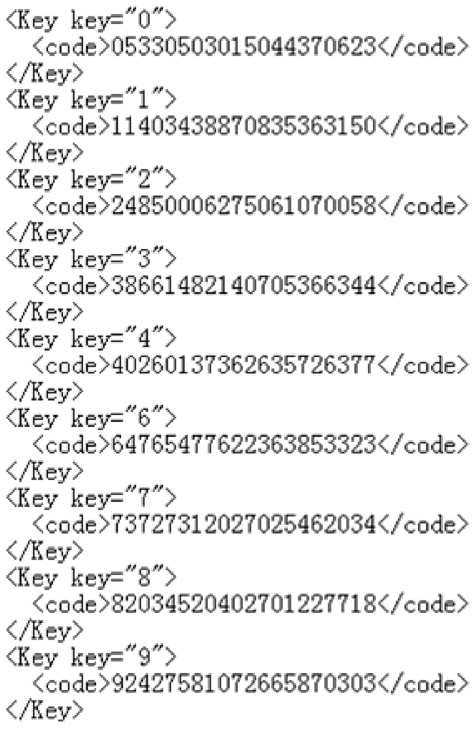
Server and Secure World.



```
 Tan code is :
51078218532423454525
```

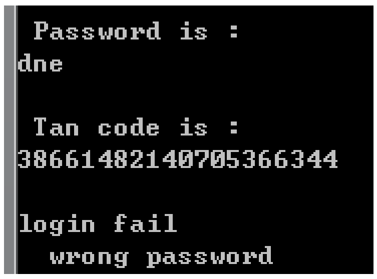


```
<Key key="0">
  <code>05330503015044370623</code>
</Key>
<Key key="1">
  <code>11403438870835363150</code>
</Key>
<Key key="2">
  <code>24850006275061070058</code>
</Key>
<Key key="3">
  <code>38661482140705366344</code>
</Key>
<Key key="4">
  <code>40260137362635726377</code>
</Key>
<Key key="5">
  <code>51078218532423454525</code>
</Key>
<Key key="6">
  <code>64765477622363853323</code>
</Key>
<Key key="7">
  <code>73727312027025462034</code>
</Key>
<Key key="8">
  <code>82034520402701227718</code>
</Key>
<Key key="9">
  <code>92427581072665870303</code>
</Key>
```



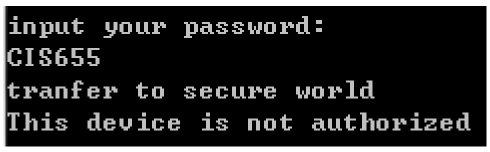
```
<Key key="0">
  <code>05330503015044370623</code>
</Key>
<Key key="1">
  <code>11403438870835363150</code>
</Key>
<Key key="2">
  <code>24850006275061070058</code>
</Key>
<Key key="3">
  <code>38661482140705366344</code>
</Key>
<Key key="4">
  <code>40260137362635726377</code>
</Key>
<Key key="6">
  <code>64765477622363853323</code>
</Key>
<Key key="7">
  <code>73727312027025462034</code>
</Key>
<Key key="8">
  <code>82034520402701227718</code>
</Key>
<Key key="9">
  <code>92427581072665870303</code>
</Key>
```

- If the password is wrong, even the TAN is correct, user still cannot login



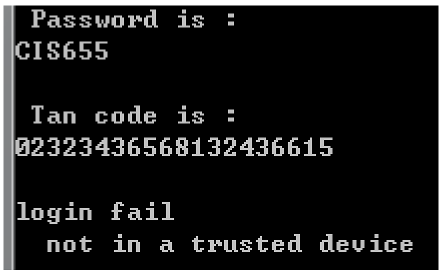
```
 Password is :
dne

 Tan code is :
38661482140705366344

login fail
  wrong password
```

- If one knows a customer's password but try to login without the customer's authorized phone, he won't make it. To test, if we delete the TAN list in Secure World (which mean others' device don't have the copy of TAN list) or change all the TAN in TAN list in Secure World (TAN is unique, all customers have different TAN list), the system won't let user to pass the login.
  - Without TAN list in Secure World



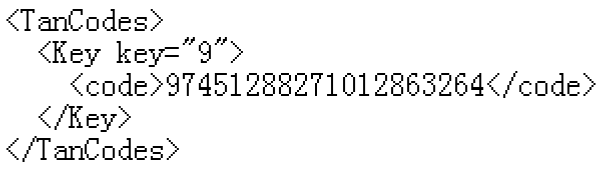
```
input your password:
CIS655
tranfer to secure world
This device is not authorized
```

- With a wrong TAN



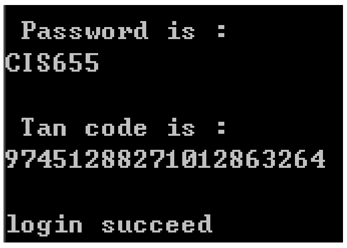
```
 Password is :
CIS655

 Tan code is :
02323436568132436615

login fail
  not in a trusted device
```

- With every customer's successful login, both the same TAN are deleted from TAN list. If after a deletion the TAN list is empty, the system will create a new one and also give a copy to customer's Secure World.
  - Only one TAN left in TAN list



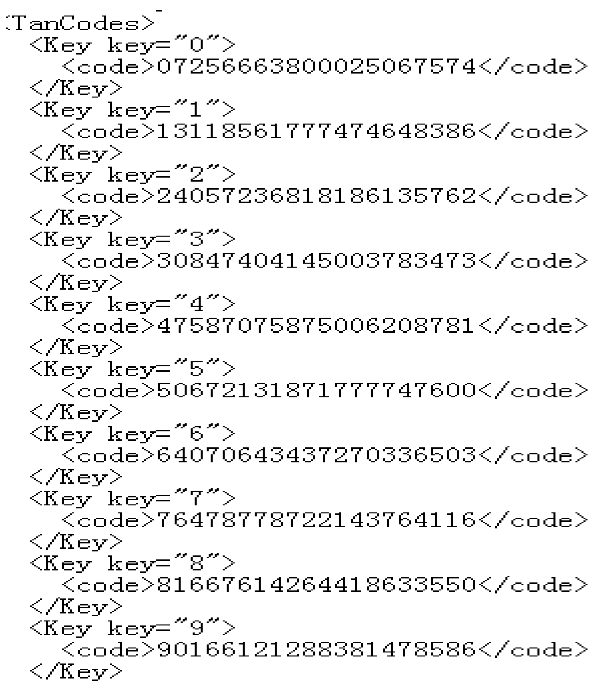
```
<TanCodes>
  <Key key="9">
    <code>97451288271012863264</code>
  </Key>
</TanCodes>
```

  - After one more successful login



```
 Password is :
CIS655

 Tan code is :
97451288271012863264

login succeed
```

  - A new list TAN list is created successfully



```
:TanCodes>
  <Key key="0">
    <code>07256663800025067574</code>
  </Key>
  <Key key="1">
    <code>13118561777474648386</code>
  </Key>
  <Key key="2">
    <code>24057236818186135762</code>
  </Key>
  <Key key="3">
    <code>30847404145003783473</code>
  </Key>
  <Key key="4">
    <code>47587075875006208781</code>
  </Key>
  <Key key="5">
    <code>50672131871777747600</code>
  </Key>
  <Key key="6">
    <code>64070643437270336503</code>
  </Key>
  <Key key="7">
    <code>76478778722143764116</code>
  </Key>
  <Key key="8">
    <code>81667614264418633550</code>
  </Key>
  <Key key="9">
    <code>90166121288381478586</code>
  </Key>
```

5.7 Conclusion

This One Time Password System provides a better security for customers' banking activities. It's a good real world use case for Trust Zone. Trust Zone provides us User Authentication, Secure Mobile Transaction, Access Control to Sensitive Data. With Trust Zone mechanism, the security of others' systems would also be improved.

Reference:

1, http://www.arm.com/zh/products/processors/technologies/trustzone/
2, http://www.openvirtualization.org/open-source-arm-trustzone.html#TrustZone-1
3,
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0333h/Chdfjdgi.html
4, http://blog.sina.com.cn/s/blog_4ce016230102v7xs.html
5,
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.kui0098a/armccref_ch04s01s13.html