ATLANTIS PRESS
Advances in Intelligent Systems Research (AISR), volume 150
2017 International Seminar on Artificial Intelligence, Networking and Information Technology (ANIT 2017)

Simulation and Realization of AT91 Microcontroller Emulator

Yongchao Tao[1, a] Wencheng Xiang[1, b] Xianghu Wu[1, c]

[1] Shenzhen Academy of Aerospace Technology, Shenzhen, China

[a]taoyongchao652@163.com, [b]HPEC_SAAT@163.com, [c]wxh_hit@126.com

Keywords: AT91 Simulation, microcontrollers, bus and equipment

Abstract. This paper focus on the designation and implementation of the AT91 simulator .which is part of the virtual verification module of the validation tools. First of all we study and analysis of the basic structure of AT91 microcontrollers. We study the bus system, address mapping, and boot model; and the logical of the external bus interface device, advanced interrupt controller's device. Then, based on the study and analysis of the AT91 microcontrollers, we abstract the overall structure of the AT91 microcontrollers, and design the basic structure of the bus and equipment as well as the standard interfaces.

1 Introduction

The AT91 family of microcontrollers is a 16/32 bit microcontroller based on the ARM7TDMI embedded microprocessor[1]. AT91 series of microcontroller contain ARM7TDMI core, embedded ICE interface, memory and peripheral components. The series has two main buses: Advanced System Bus (Advanced System Bus) and Advanced Peripheral Bus (Advanced Peripheral Bus). The ARM7TDMI implements interconnection of on-chip 32-bit memory, external bus interface EBI and AMBA Bridge through the ASB interface. AMBA Bridge is used to drive APB. APB is used to access the on-chip peripherals and optimize system power consumption. Series products have external bus interface EBI[2]. With it, the ARM core can be connected to external memory and dedicated peripherals. EBI supports 8-bit and 16-bit devices, and can use two 8-bit devices to simulate a 16-bit device.

The AT91 series incorporates on-chip peripherals, including external bus interfaces, advanced interrupt controllers, parallel I/O controllers, general purpose synchronous/asynchronous transceivers, timers, and memory[3]. AT91 microcontrollers to ATMEL high-density CMOS manufacturing process, through the ARM7TDMI processor core and high-speed on-chip memory and a variety of peripheral function modules integrated in a single silicon, which make the AT91 microcontroller for a variety of large amounts of embedded applications need to provide a flexible, cost-effective solution[4].

According to AT91 microcontrollers' hardware structure and logic functions, this article will perform instruction-level emulation for its bus, memory, and on-chip devices[5]. And we will define the bus interface function, data structure, device interface function, data structure, external device standard interface function.

2 AT91 Microcontroller Architecture Abstract

The structure of the AT91 microcontroller is abstracted into three parts: 1) ARM7TDMI core: AT91 microcontroller microprocessor core is ARM7TDMI microprocessor, which is the core of the microcontroller components which controls the operation of the entire microcontroller. From the AT91 microcontroller overall, ARM7TDMI core interacts with peripherals and memory mainly through the bus. When a read and write operation is required, the address bus is addressed to the corresponding device or memory and the data is exchanged via the data bus.

2) AT91 microcontroller has two main buses: ASB (Advanced System Bus): The ARM7TDMI implements interconnection of on-chip 32-bit memory, external bus interface EBI, and AMBA bridge through the ASB interface. AMBA Bridge is used to drive APB. APB (Advanced Peripheral

Bus): APB bus is mainly used to access the on-chip peripherals, optimize system power consumption. As the system is instruction set simulation, the other AMBA bridge for the ARM7TDMI core is transparent, so the simulation can be ignored, so that you can use a bus to simulate the entire bus structure, and greatly simplifies the simulation to achieve Complexity, while improving the stability of the system, reducing the possibility of error.

3) Device (including memory): The AT91 microcontroller contains a large number of on-chip devices and on-chip memory. They interact with the ARM7TDMI core via the bus. Each device or memory has an address space and has its own functional logic. The bus is addressed to the corresponding device via the address bus and read and writes via the data bus. So each device can be abstracted into one independent device object. Each device object has an address space, that has a starting address attribute, an address space size attribute, has a piece of its own storage space, this space can record the device's real-time status. In addition, each device should have an interface function set, the bus through these interface functions to access and operate the device.

Based on the above analysis, we can abstract all the devices into the same structure of the object. They are through the bus and ARM7TDMI core to interact. So the entire AT91 microcontroller is abstracted into a three-tier structure: ARM core, bus, and device. The hierarchical structure is shown in Figure 1.

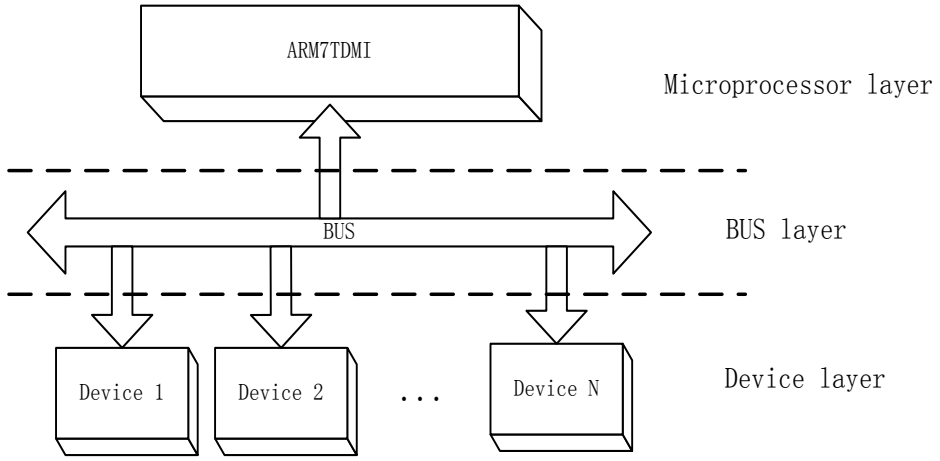

Fig.1. AT91 Microcontroller Abstraction Hierarchy

So the AT91 microcontroller simulation implementation, the AT91 microcontroller will also be divided into three parts according to the three-tier structure.

1) Simulation core: simulation core mainly simulate the microprocessor instruction execution process, which through the bus manager to provide the interface function to read and write external devices.

2) Bus: the device management, address mapping and addressing, for the simulation kernel to provide data access interface function.

3) Device: to achieve the function of each device simulation, to provide external data access interface, which mainly emulates on-chip memory, external bus manager EBI and advanced interrupt controller AIC and other devices.

3 AT91 simulation internal structure definition

Bus structure definition: The main function of the bus is to manage the device, address mapping, decoding addressing and other operations. So it needs to have a device table for storing all the devices it manages, also needs an address mapping table, each device will be mapped to a different address up, and when the implementation of remapping, you can modify the address mapping table to change the address mapping, and what's more it needs to have an address bus, the data bus used to temporarily store the data to be interactive. In addition, according to the overall design of the simulation system, the bus structure also needs to save a device interface function set and a configuration file pointer for the initialization phase of the device configuration. So the structure of

the bus is defined as follows:

```
Struct  Bus  {
        Unsigned long addr;          // Analog address bus, save the physical address
        Unsigned long data;          // Analog data bus, save data
        Unsigned char map_table[4096]; // Address mapping table, used to manage the 4G's physical address space
        Device* DeviceTable;         // The device set by the bus management
        Func* func_set;              // All sets of interface functions for the device
        FILE* Configure_fp;          // The configuration file pointer for the microcontroller
}
```

Device structure definition: All devices have three common attributes: address range, memory space, and interface function. And through these three attributes we have been able to fully identify a device. So all devices can be defined as the same structure, as follows:

```
Struct Device  {
   Unsigned long start_addr;   // The starting address of the device
   Unsigned long size;         // The address space of the device
   Unsigned char * mm;         // The device has a memory space that contains all user-visible registers and user-invisible registers, as well as some other global variables, cache, etc.
   Func* funcptr;              // Device interface function set, when the device mount a different interface function set, it represents a different device.
}
```

4 Address decoding algorithm

As the operation of the program requires a lot of imitation operation, so the simulator's imitation speed is a key factor affecting the efficiency of the simulator. As the AT91 microcontroller has a lot of equipment and memory, so how to improve the efficiency of address decoding, has become the key to the impact of imitation speed. According to the characteristics of AT91 address space allocation. This paper adopts a discrete addressing algorithm to realize a fast address decoding algorithm. Using this algorithm can greatly improve the efficiency of address decoding, fast positioning equipment, thereby improving the speed of the entire simulator. The algorithm idea is as follows:

This article will divide the 4G size of the memory space by blocking the size of 1M(4K = 4096 blocks). We will use the array with 4096 elements map_table management of each block, each element in the array contains the address space mapped to the device of the device number, 0 that the corresponding 1M address of the block is an empty device, as shown in the following figure 2 shows:

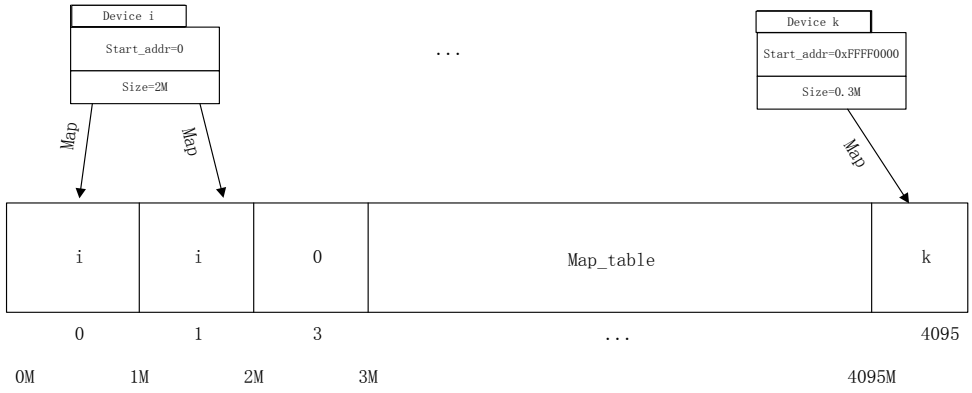

Fig.2. Address mapping diagram

When adding a new device, according to the device's starting address and address space size, its device number mapping to map_table, the mapping method is:

```
Mapping(int  i)  {
        Start= The starting address of device i >>20;
        End=（The starting address of device i +size）>>20;
        For(j=start;j<=end;j++)  {
            k=map_table[j];
            If(k!=0  &&  Device k is not a virtual device)      {
                Create a virtual device w;
                Generates 64 bytes of memory for the virtual device w;
                Addvirmap（&Bus_Ctr , j , i）;
            }
            Else if（k!=0）  {
                Addvirmap（&Bus_Ctr , j , i）;
            }
            Else
                Map_table[j]=i;
        }
}
```

The mapping process is shown in Figure 3. After setting up the address mapping table map_table, if you give a physical address, you can quickly find the device number that may contain this physical address, the algorithm is as follows:

1) Get the physical address addr.

2) Retrieves the device number of the device containing the physical address from the map_table:Devnum = map_table [Addr >> 20]

3) Call the function of this device: Device_Table [Devnum] .funcptr [i].

5 Virtual device

When two or more devices are mapped to the same 1M address space, it will lead to mapping conflicts, resulting in addressing errors. To eliminate this conflict, we define a virtual device, which has a 64-byte mapping table that divides the 1M space into 64 blocks of 16KB each. Virtual devices use this mapping table to manage devices in the same 1M space.

When a 1M space corresponds to multiple devices, we create a virtual space for this 1M space. According to the starting address and size of each device, we map it to the mapping table owned by this virtual device and fill in the device number of the virtual device in the map_table entry corresponding to this 1M space. After the mapping is complete, each item in the mapping table records the device number of the device mapped to the address space it manages. When the address of this 1M space to decode, we first from the map_table table to find the virtual device number, And then through the virtual device owned by the mapping table ,we can find the corresponding device number, and then from the equipment table to find the appropriate equipment.

6 Test results

The emulator is finally integrated with the arm core simulator and provided in the form of a DLL to the user and linked with the changed GDB before it can be simulated. And, in order to interface friendly, the application has been developed a good VC-based front-end procedures, through the front-end program for GDB and simulation kernel DLL link and test. Test the host computer is based on the Windows system x86 machines, directly run by the VC development of the front-end procedures, through the link simulator operation, you can complete the GDB and simulation of the nuclear link, then, you can test validation.

This emulator requires several functional tests: 1) information acquisition function, AT91 simulator is able to return the correct register and other information.2) the basic simulation function, the preparation of a program, the implementation of continuous simulation, and view the registers, memory, etc., to see whether the implementation of the correct. 3) Interrupt function test, trigger an external interrupt, to see if the response.

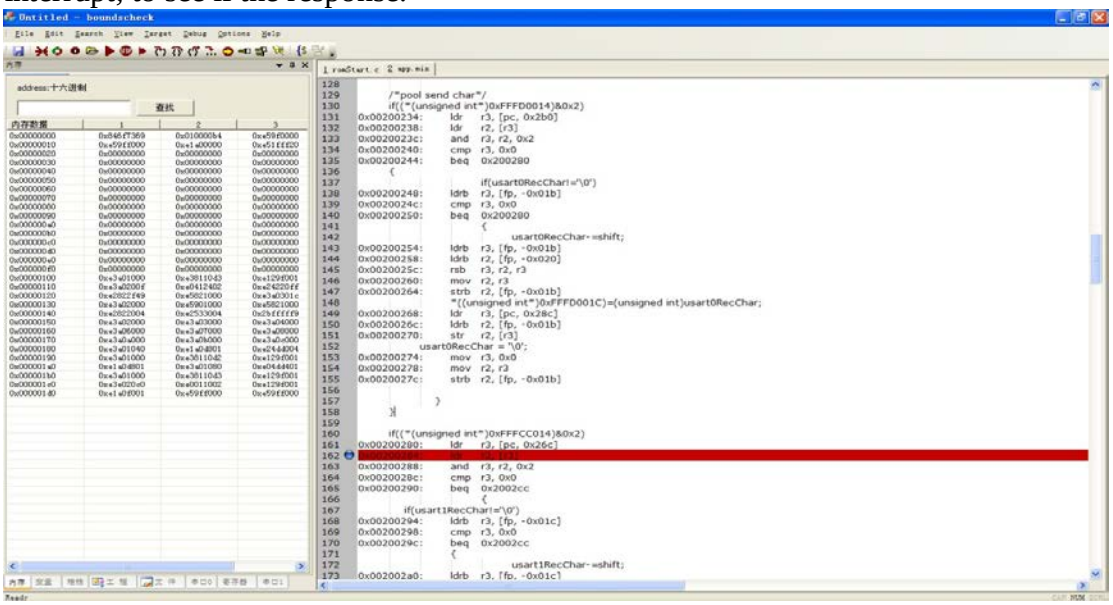

Fig.3. Function verification interface

Through the above test of the simulator, we can prove that the AT91 microcontroller emulator can be correctly integrated with GDB, and can correctly simulate the AT91 microcontroller's functional logic, so that the executable program can run on it correctly.

7 Conclusion

This paper mainly studies the simulation of AT91 micro controller emulator. First of all, the overall structure and frame of AT91 microcontrollers are abstracted and abstracted into three layers, CPU layer, bus layer and device layer. Then in this paper, an object-oriented method is used to define a set of standard description structure and interface functions for the device.

Acknowledgement

In this paper, the research was sponsored by the future industry special funds of Shenzhen (Project No. CXZZ20140718154349249).

References

[1] Julian T.Brown. ARMphetamine A Dynamically Recompiling ARM Emulator. Queens'College, 2000:1~10

[2] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. Introduction To Automata Theory, Languages And Computation. second edition. Addison Wesley, 2001: 316~327

[3] Asad K, Weiqiang M, Chris W, et al. Multi-Threaded Simics SystemC Virtual Platform[C], ICC Atmel Corporation. AT91 Assembler Code Startup Sequence for C Code Applications Software. DOC2644. 2002: 1~16

[4] Atmel Corporation. Interrupt Management: Auto-vectoring and Prioritization. DOC1168.

[5] Atmel Corporation. AT91 Reset Considerations. DOC2645. 2002: 1~17