QEMU를 활용한 at91sam9xe(ARM926EJ-S) processor simulator 구현

*윤동준, 유호영, 조지혁, 박인철
한국과학기술원 전기 및 전자공학과
e-mail : *djyoon.ics@google.com, hyyoo.ics@google.com, jhjo.ics@google.com, icpark@kaist.edu*

Implementation of at91sam9xe(ARM926EJ-S) processor simulator Using QEMU

*Dong-Joon Yoon, Ho-Young Yoo, Ji-Hyuk Jo, In-Cheol Park
School of Electrical Engineering
KAIST

Abstract

Computer architects have traditionally evaluated instruction sets and microarchitectures by using in-house simulator. Making in-house simulator have large burden as much as designing processor. So, Using QEMU that already proved as open source emulator has many benefits than making in-house simulator. We modified QEMU to use as at91sam9xe processor simulator at a small expenditure of effort.

I. 서론

일반적으로 processor 설계자들은 설계한 processor의 instruction set과 반영한 architecture의 성능을 검증하기 위해 자체 개발한 simulator를 이용해왔다. 자체적으로 simulator를 개발하는 일은 설계한 instruction set을 software로 구현하는 일 뿐만 아니라 simulator가 잘 동작하게 하기위한 환경구성 및 구현한 simulator의 검증까지 생각해야 하므로 processor 설계만큼이나 시간을 써야하는 일이다. 예를 들어 일반적으로 많이 사용되는 embedded processor인 ARM의 simulator를 구현한다고 하면, 순수하게 ARM architecture의 구현에만 집중한다 하더라도 ARM의 37개 general purpose register들의 instruction 수행에 따른 data 변화 및 7개의 mode(usr, system, supervisor, ..) 전환에 의한 register bank의 switching, Memory Subsystem(MMU, Cache) 과 Copressor의 operation behaviour를 전부 고려해야 한다. 따라서 본 논문에서는 공개적으로 구현되고 검증된 open source emulator인 QEMU를 활용해 검증하고자 하는 processor의 simulation에 사용하는 방법에 대해 설명한다. QEMU를 이용해 simulation 하려는 processor는 QEMU에서 지원하는 ARM architecture 중 QEMU의 emulation list에는 들어 있지 않는 Ateml사의 AT91SAM9XE를 사용한다. 본 논문의 나머지 부분인 Ⅱ. 본론에서는 QEMU에 대한 간략한 설명과(2.1), 실제로 AT91SAM9XE를 어떻게 porting하였는지에 대한 방법을 설명한다(2.2). 또한 추가된 architecture위에 linux를 porting해 간단한 application를 수행해봄으로서 AT91SAM9XE processor가 잘 emulating 되었는지 판단한다(2.3). Ⅲ. 구현에서는 Ⅱ. 본론에서 제시한 방법에 대한 구현 결과를 보여주고, Ⅳ. 결론 및 향후 연구 방향에서는 Ⅰ, Ⅱ, Ⅲ에서 제시하였던 방법과 결과에 대한 마무리를 짓는다.

II. 본론

2.1 QEMU: Dynamic translator

QEMU는 Quick EMUlator의 약자로 2005년 Fabrice bellard가 발표한 open source system emulator이다. QEMU는 크게 2개의 부분으로 나눌 수 있는데, 첫 번째는 target machine의 binary가 host machine에서 실행될 수 있도록 binary를 translation해주는 부분과 두 번째는 target machine의 device들을 emualtion하여 host machine의 device들과 연결될 수 있게 하는 부분이다[1].

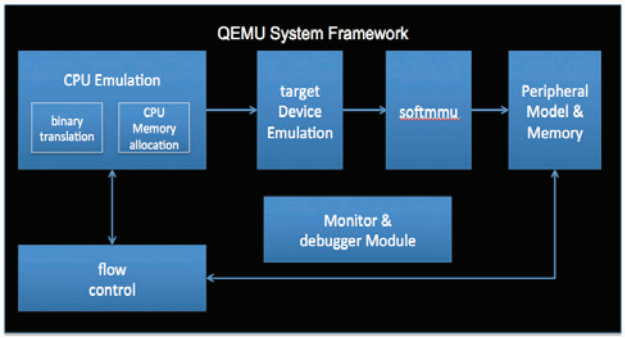

그림 1. QEMU system framework

QEMU의 binary translation 방식은 dynamic translation 방식으로, target machine의 binary를 매번 해석(intepretion)하여 실행하는 intepretor 방식과 차이가 있다. intepretor 방식은 target의 수행코드를 host의 수행코드로 변경하는 작업을 매번 수행하므로, target system의 clock이 높을 경우, 실시간 성능을 만족시키기 어렵다. 반면에 dynamic translation 방식은 translation의 단위가 target의 branch instruction이나 processor의 상태정보가 바뀌는 instruction들에 의하여 나누어진 block이므로 매 instruction마다 수행코드를 변경해야 하는 overhead가 없고, 또한 translation된 block을 translation cache라는 곳에 저장하여, 이미 translation된 block이 다시 수행될 경우에 translation 과정없이 바로 수행될 수 있도록 한다. 이러한 dynamic translation 방식을 사용함으로써 QEMU는 일반적으로 사용되는 다른 system simualtor급 이상(simics, SESC등)의 수행 속도를 제공한다.

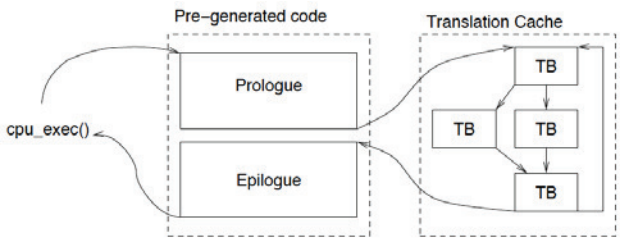

그림 2. Dynamic translation of QEMU

그림 2는 QEMU의 instruction translation을 나타낸다. QEMU는 target instruction의 block을 동적으로 translation하고 그림 2의 오른쪽 부분처럼 이미 translation된 block을 chaining방식으로 관리한다.

현재 QEMU는 1.1 version까지 나와있고, 가장 최근의 stable한 version은 1.0.1 version이다. 본 논문에서 사용한 1.0.1의 QEMU에서는 target processor로 x86, ARM, MIPS, SPARC등을 지원하고, linux, window같은 운영체제를 수정 없이 QEMU 위에서 구동할 수 있다. 그림 3은 QEMU 1.0.1 version의 directory tree를 간단히 표현한 것이다.

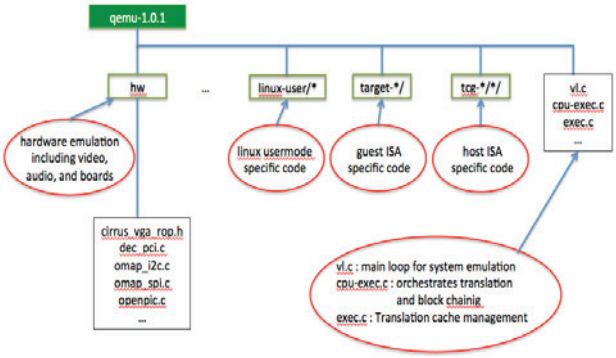

그림 3. QEMU source directory tree

2.2 Porting AT91SAM9XE on QEMU

AT91SAM9XE는 atmel에서 만든 ARM926EJ-S core의 processor로서 최대 180Mhz의 clock에서 구동가능하고 8Kbyte의 Data Cache, 16Kbyte의 Instruction Cache, Write Buffer, MMU를 가지고 있다. 그 외 여러 가지의 On-chip peripheral controller들을 가지고 있는데, 대표적인 것들만 보자면 USART, EMAC, USB, SPI, AIC들이 있다. 좀 더 자세한 processor 구조에 대한 것은 아래의 그림 4을 참조한다.

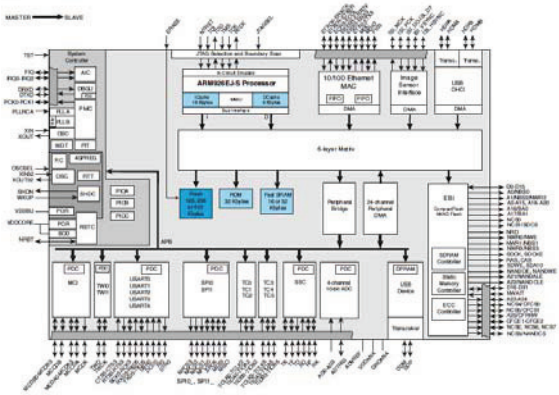

그림 4. AT91SAM9XE block diagram

QEMU에 AT91SAM9XE processor를 추가하기 위해선 일단, AT91SAM9XE를 QEMU를 system

emulating list에 추가하여야 한다. 그러기 위해선 QEMU의 Makefile.target파일에 at91sam9xe processor core와 on-chip peripheral controller들을 target으로 추가한다(obj-arm-y+=at91sam9xe.o at91sam9_usart.o at91sam9_emac.o ...). 그리고 AT91SAM9XE의 전체적인 system 초기화(core momery initialzation peripheral 등록 및 initialzation)에 관한 내용을 processor core를 모사한 파일에 구현하고 (at91sam9xe.c), processor core를 모사한 파일에서 사용하는 peripheral controller들의 기능들은 각각 하나의 파일로 모사한다(at91sam_usart.c, at91sam_emac.c ...). 이 때 해당 peripheral을 모사하는 파일에는 core가 peripheral에 접근할 때 사용하는 read/write함수(handler)가 반드시 포함되어 있어야한다.

QEMU에서는 processor core를 qemu의 지원 machine list에 등록하고, peripheral들을 sysbus 라는 일종의 main system bus에 등록을 해 주어야 한다. 먼저 processor core를 초기화하고 machine list에 등록하는 과정부터 살펴보면, 아래와 같은 그림 5로 표현할 수 있다.

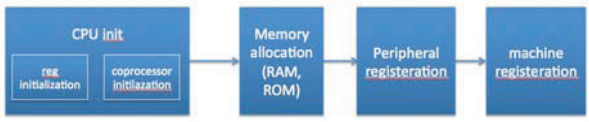

그림 5. processor core initialization

각 processor의 cpu_model(arm926, arm946, coretex-a8 ...)에 맞게 core의 register value들이 초기화되고, coprocessor들이 생성한다. 그리고 core에서 사용할 memory(RAM, ROM)가 할당되고, on-chip peripheral controller들이 sysbus에 등록된다. core와 peripheral들이 전부 초기화 된 후 processor는 machine-list에 등록된다. 따라서 AT91SAM9XE를 machine list에 등록하기 위해서는 cpu_model을 arm926EJ-S로 지정하여 CPU를 초기화 시켜주고, AT91SAM9XE에서 사용하는 memory를 그림 6의 memory map을 참조하여 할당시켜 주어야한다.

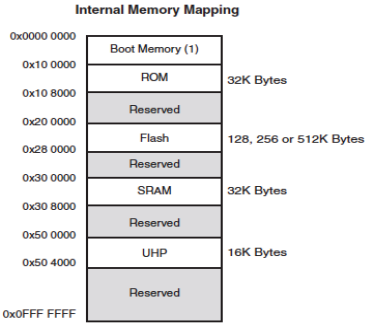

그림 6. AT91SAM9XE internal memory map

CPU 초기화와 memory 할당이 다 끝난 후엔 AT91SAM9XE에서 사용한 peripheral들을 sysbus에 등록시켜준 후, AT91SAM9XE를 machine list에 등록한다. 아래의 그림 7은 processor core의 초기화와 machine등록 source code의 일부분이다.

```
memory_region_init_ram(ram, NULL, "at91sam9xeek.ram", AT91SAM9XE_SDRAM_SIZE);
memory_region_add_subregion(address_space_mem, AT91SAM9XE_SDRAM_BASE, ram);
//memory_region_init_alias(ram_alias, "ram.alias", ram, 0, AT91SAM9XE_SDRAM_SIZE);
//memory_region_add_subregion(address_space_mem, AT91SAM9XE_SDRAM_BASE, ram_alias);

memory_region_init_ram(flash, NULL, "at91sam9xeek.nand", AT91SAM9XE_NAND_FLASH_SIZE);
memory_region_add_subregion(address_space_mem, AT91SAM9XE_NAND_FLASH_BASE, flash);

cpu_pic = arm_pic_init_cpu(env);
dev = sysbus_create_varargs("at91sam9_aic", AT91SAM9XE_AIC_BASE, cpu_pic[ARM_PIC_CPU_IRQ],
                            cpu_pic[ARM_PIC_CPU_FIQ], NULL);

for (i = 0; i < 32; i++) {
    pic[i] = qdev_get_gpio_in(dev, i);
}

// system peripheral interrupt
dev = sysbus_create_simple("at91sam9_intor", -1, pic[1]);
for (i = 0; i < 32; i++) {
    pic1[i] = qdev_get_gpio_in(dev, i);
}
sysbus_create_simple("at91sam9_dbgu", AT91SAM9XE_DBGU_BASE, pic1[0]);
pmc = sysbus_create_simple("at91sam9_pmc", AT91SAM9XE_PMC_BASE, pic1[1]);
qdev_prop_set_uint32(pmc, "mo_freq", 16000000);
sysbus_create_simple("at91sam9_pit", AT91SAM9XE_PIT_BASE, pic1[3]);
sysbus_create_simple("at91sam9_rstc", AT91SAM9XE_RSTC_BASE, pic1[4]);
sysbus_create_simple("at91sam9_rtt", AT91SAM9XE_RTT_BASE, pic1[5]);

// user peripheral interrupt
//sysbus_create_varargs("at91sam9_tc", AT91SAM9XE_TC012_BASE, pic[19], pic[19], pic[19], NULL);
sysbus_create_varargs("at91sam9_tc", AT91SAM9XE_TC012_BASE, pic[17], pic[18], pic[19], NULL);
sysbus_create_simple("at91sam9_spi", AT91SAM9XE_SPI0_BASE, pic[12]);
sysbus_create_simple("at91sam9_spi", AT91SAM9XE_SPI1_BASE, pic[13]);
sysbus_create_simple("at91sam9_pio", AT91SAM9XE_PIOA_BASE, pic[2]);
sysbus_create_simple("at91sam9_pio", AT91SAM9XE_PIOB_BASE, pic[3]);
sysbus_create_simple("at91sam9_pio", AT91SAM9XE_PIOC_BASE, pic[4]);

qemu_check_nic_model(&nd_table[0], "at91");
sysbus_create_simple("at91sam9_emac", AT91SAM9XE_EMAC_BASE, pic[21]);
```

그림 7. processor core initialization source code

두 번째로 QEMU에서 peripheral들을 sysbus에 등록하는 방법을 살펴보면, 전체적인 과정을 아래의 그림 8로 표현할 수 있다.

그림 8. peripheral initialization

해당 peripheral을 sysbus에 creation하는 함수를 호출하여 peripheral을 main system bus에 등록하는데, 이 때 sysbus creation함수에 peripheral이 mapping될 memory의 address와 interrupt number를 인자로 전달한다. sysbus creation 함수에서는 인자로 전달받은 interrupt number를 기반으로 해당 peripheral을 core에 interrupt를 줄 수 있는 IRQ device로 등록한다. 그리고 해당 peripheral을 모사한 파일에 기술되어 있는 read/write함수를 core가 sysbus의 해당 peripheral을 접근하려 할 때 사용되는 handler로 등록된다.

AT91SAM9XE processor를 추가하기 위한 전체 과정을 요약하면, core가 사용하는 각각의 peripheral들이 해당 device들을 위한 read/write 함수(handler)들과 같이 등록이 되어 있으면, 등록이 되어 있는 peripheral들을 core의 정보와 core가 사용할 memory를 초기화 할 때 같이 초기화하여 사용할 수 있게 준비한 다음, 해당 processor를 machine list에 등록한다.

2.3. Linux Porting emulated AT91SAM9XE

QEMU에 추가된 AT91SAM9XE 위에 linux를 porting하는 과정은 일반적인 embedded linux porting과정과 같다. target으로 하는 platform을 반영한 configuration으로 compile된 linux 커널을 커널이 사용할 파일시스템과 같이 porting하는 것이다. 따라서 사용할 2.6.30 version의 linux kernel을 default로 제공하는 at91sam9xeek_defconfig configuration파일을 이용하여 compile을 한 후 busybox를 이용해 만든 파일 시스템과 같이 porting한다. linux 커널과 파일 시스템을 porting 한 후, porting이 제대로 이루어졌는지를 확인하기 위하여, GNU arm cross compiler를 이용하여 만든 간단한 logo print application을 porting된 linux 커널 위에서 실행하여 본다. 아래의 그림 9는 logo print application의 수행결과이다.

그림 9. running logo print application on linux

Ⅲ. 구현

아래의 그림 10은 QEMU에 AT91SAM9XE를 추가하여 QEMU를 구동시킨 후 QEMU의 command 창에서 현재의 memory map과 추가된 device등을 확인한 그림이다.

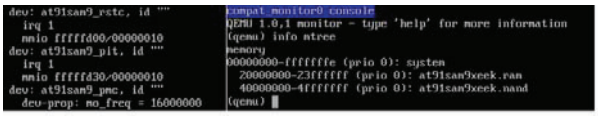

그림 10. QEMU memory, device tree

QEMU에서 구동되는 AT91SAM9XE에 porting된 리눅스 커널위에서 cpu의 information과 memory의 information을 확인한 결과는 아래의 그림 11와 같다.

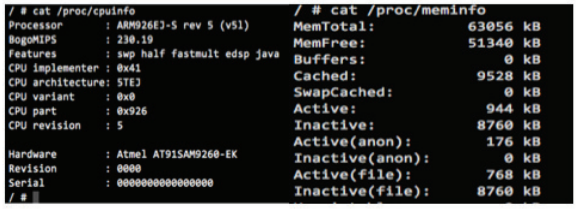

그림 11. cpu, memory information of embedded linux

cpu information에 arm926ej-s core와 atmel at91sam9xeek(at91sam9290-ek와 동일)의 hardware가 잘 표시됨을 확인할 수 있다.

Ⅳ. 결론 및 향후 연구 방향

본 논문에서는 processor simualtor로 QEMU를 활용하는 방법에 대해 알아보았으며, QEMU에 추가된 target processor위에 리눅스와 파일 시스템을 porting하고 간단한 logo print application을 구동해보았다. 향후 ARM instruction set architecture compatible한 processor의 simulator가 필요할 때 본 논문에 제시한 방법이 유용할 것이라 생각되며, ARM과 관련되어 있지 않은 새로운 architecture라 할지라도 본 논문에 제시된 방법의 flow를 따라가면 어렵지 않게 QEMU를 새로운 architecture의 simulator로 활용할 수 있을 것이라 기대한다.

Acknowledgement

이 논문은 정부(교육과학기술부)의 재원으로 (재)다차원 스마트 IT 융합 시스템 연구단(글로벌프론티어사업)의 지원을 받아 수행된 연구임 ((재)스마트 IT 융합 시스템 연구단-No.20110031860)

참고문헌

1] Fabrice Bellard, “Qemu, a fast and portable dynamic translator,” USENIX 2005 annual technical reference 2005년 4월

[2] 최종욱, 남병규, “탑재소프트웨어 개발을 위한 QEMU 기반의 고성능 LEON3 프로세서 에뮬레이터 개발,” 항공우주학회논문지, 1016-1021쪽, 2012년 4월

[3] www.qemu.org