TI DSP,MCU 및 Xilinux Zynq FPGA 프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/5/2
수업일수	46 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

coretex M4 부팅코드 살펴보기

어떤 CPU 든 부팅 할 때 초기화를 함. POR(post on reset)가 실행되고 reset_handler 를 발생함.

ex) 아두이노 빨간버튼 - 인터럽트(리셋핸들러)

-reset_handler

```
; Reset handler
Reset_Handler   PROC
                EXPORT  Reset_Handler             [WEAK]
        IMPORT  SystemInit
        IMPORT  __main

                ;FPU settings
                LDR     R0, =0xE000ED88         ; Enable CP10,CP11
                LDR     R1,[R0]
                ORR     R1,R1,#(0xF << 20)
                STR     R1,[R0]

                LDR     R0, =SystemInit
                BLX     R0
                LDR     R0, =__main
                BX      R0
                ENDP

; Dummy Exception Handlers (infinite loops which can be modified)

NMI_Handler     PROC
                EXPORT  NMI_Handler               [WEAK]
                B       .
                ENDP
HardFault_Handler₩
                PROC
                EXPORT  HardFault_Handler         [WEAK]
                B       .
```

R0 레지스터에 0xE000ED88 를 넣음.

R1 에 R0(0XE000ED88 주소)가 가리키는 값을 넣음.

ORR 1111(2)<<20 와 R1 과 OR 연산한 값을 r1 에 넣음.
20,21,22,23 비트를 1 로 세팅하겠다.

R0 레지스터에 SystemInit 을 세팅.

BLX - 함수 호출해서 R0 에 들어있는 systeminit 이 호출된다.

[Cortex-M4 Technical Reference Manual - 4.2 Register summary]

Address 0xE000ED88

Name CPACR
Type RW
Reset -
Description Coprocessor Access Control Register

arm 에는 MMU 를 제어하기위한 별도의 하드웨어 CP(coprocessor)가 존재. CP0 부터 CP15 까지의 레지스터를 가지는데 각각이 프로세서의 동작을 서포트 해줌. CP10, CP11 은 부동소수점을 사용할 수 있도록 함.

[Cortex-M4 Device Generic User Guide - 4.6.1 Coprocessor Access Control Register]

CPACR 레지스터는 보조 프로세서에 대한 엑세스 권한을 지정.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								CP11		CP10		Reserved																			

Function

Access privileges for coprocessor n. The possible values of each field are: 0b00 = Access denied. Any attempted access generates a NOCP UsageFault 0b01 = Privileged access only. An unprivileged access generates a NOCP fault. 0b10 = Reserved. The result of any access is Unpredictable. 0b11 = Full access.

Bit	Name	Function
31~24	-	
19~0	-	예약(Reserved)
23, 22	CP11	0bXX
21, 20	CP10	0bXX

CP11, CP10 - 부동소수점을 표현할 수 있도록 하는 레지스터인데, CP11, CP10 둘다 활성화 해야 부동 소수점을 표현할 수 있다.

‘**ORR R1, R1,#(0xF << 20)**’ 명령어에서 CP11, CP10 를 0b11 로 활성화 하여 Full access 의 권한을 얻게 됨.

- SystemInit

coretex M4 는 32 비트시스템을 사용.

```
/**
  * @brief  Micro Controller System을 설정한다.
  *         Embedded Flash Interface, PLL을 초기화하고 SystemFrequency 변수를 갱신한다.
  * @param  None
  * @retval None
  */
void SystemInit(void)
{
  /* RCC Clock 구성을 Default Reset State로 reset(재설정)한다. */
  /* Set HSION bit */
  RCC->CR |= (uint32_t)0x00000001;

  /* Reset CFGR register */
  RCC->CFGR = 0x00000000;

  /* Reset HSEON, CSSON and PLLON bits */
  RCC->CR &= (uint32_t)0xFEF6FFFF;

  /* Reset PLLCFGR register */
  RCC->PLLCFGR = 0x24003010;

  /* Reset HSEBYP bit */
  RCC->CR &= (uint32_t)0xFFFBFFFF;

  /* 모든 Interrupt를 비활성화한다. */
  RCC->CIR = 0x00000000;

#ifdef DATA_IN_ExtSRAM
  SystemInit_ExtMemCtl();
#endif /* DATA_IN_ExtSRAM */

  /* System Clock Source, PLL 곱셈기, 나눗셈기, AHB/APBx Prescalers와 Flash 설정을 구성한다. */
  SetSysClock();

  /* Offset Address를 더한 Vector Table 위치를 구성한다. */
#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* 내부 SRAM에 Vector Table 재배치 */
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* 내부 FLASH(NAND)에 Vector Table 재배치 */
#endif
}
```

실제 System clock 은 높은데 RC oscillator 로 주파수를 만들 수 없어서 PLL 증폭을 위해 곱셈기, 나눗셈기가 들어감.

AHB/APBx 가 데이터가 흘러 들어가는 속도 값 조정

RCC : Reset and Clock Control for STM32F42xxx and Stm32F43xxx

[RM0090 Reference Manual(STM32F407) - 6.3.1 RCC clock control register (RCC_CR)]

: clock 조정하는 레지스터

Address offset: 0x00 Reset value: 0x0000 XX83 where X is undefined. Access: no wait state, word, half-word and byte access

'RCC->CR |= (unit32_t)0x00000001' => bit 0 을 1 로 세팅

Bit 0 HSION: Internal high-speed clock enable Set and cleared by software. Set by hardware to force the HSI oscillator ON when leaving the Stop or Standby mode or in case of a failure of the HSE oscillator used directly or indirectly as the system clock. This bit cannot be cleared if the HSI is used directly or indirectly as the system clock. 0: HSI oscillator OFF **1: HSI oscillator ON - RC 발진기 ON**

첫 번째 비트를 1 로 세팅하면서 HSI(High Speed Internal) 발진기를 사용함.

[RM0090 Reference Manual(STM32F407) - 6.3.3 RCC clock configuration register (RCC_CFGR)]

: clock 환경설정 레지스터, 클록을 어떤 식으로 구성 할 건지 결정하는 레지스터

' RCC->CFGR = 0x00000000 ' => 처음 전기신호가 들어가면서 다른 회로와 플로팅 될 수 있으므로 리셋 시켜 clock 환경 레지스터 초기화.

Address offset: 0x08 Reset value: 0x0000 0000 Access: 0 ≤ wait state ≤ 2, word, half-word and byte access 1 or 2 wait states inserted only if the access occurs during a clock source switch.

[RM0090 Reference Manual(STM32F407) - 6.3.1 RCC clock control register (RCC_CR)]

'RCC->CR &= (unit32_t)0xFEF6FFFF' => 24, 19, 16 번째 비트 : 0 , PLL, Clock Security System Enable, HSE Oscillator 을 리셋 시킴.

Bit 24

PLLON : Main PLL (PLL) enable
Set and cleared by software to enable PLL. Cleared by hardware when entering Stop or Stand by mode. This bit cannot be reset if PLL clock is used as the system clock.
0: PLL OFF
1: PLL ON

PLL(Phase Locked Loop) – 위상 고정 루프 : 주파수를 고정시킨다. 클록을 생성하는데 고주파를 생성.

Bit 19

CSSON : Clock security system enable
Set and cleared by software to enable the clock security system. When CSSON is set, the clock detector is enabled by hardware when the HSE oscillator is ready, and disabled by hardware if an oscillator failure is detected.
0: Clock security system OFF (Clock detector OFF)
1: Clock security system ON (Clock detector ON if HSE oscillator is stable, OFF if not)

HSE(High Speed External)

Bit 16

HSEON : HSE clock enable
Set and cleared by software. Cleared by hardware to stop the HSE oscillator when entering Stop or Standby mode. This bit cannot be reset if the HSE oscillator is useddirectly or indirectly as the system clock.
0: HSE oscillator OFF
1: HSE oscillator ON

하드웨어가 플로팅 할 수 있으므로 다 OFF 시킴.

[RM0090 Reference Manual(STM32F407) - 6.3.2 RCC PLL configuration register (RCC_PLLCFGR)]

‘ RCC->PLLCFGR = 0x24003010 ’ =>RLLCFGR 을 초기화 시킴

Address offset: 0x04 Reset value: **0x2400 3010** Access: no wait state, word, half-word and byte access.

PLL 은 고주파 클록을 발생시키는데, 여기서 플로팅 값이 들어와 증폭시키게 되면 고주파는 전류가 세지게 되어 보드가 탈 가능성이 있으므로 리셋 시킴.

[RM0090 Reference Manual(STM32F407) - 6.3.1 RCC clock control register (RCC_CR)]

‘RCC->CR &= (unit32_t)0xFFFBFFFF’ → 18 번째 비트를 0 으로 초기화 함.

Bit 18 HSEBYP : HSE clock bypass Set and cleared by software to bypass the oscillator with an external clock. The external clock must be enabled with the HSEON bit, to be used by the device. The HSEBYP bit can be written only if the HSE oscillator is disabled. **0: HSE oscillator not bypassed** 1: HSE oscillator bypassed with an external clock

[RM0090 Reference Manual(STM32F407) - 6.3.1 RCC clock interrupt register (RCC_CIR)]

‘RCC->CIR = 0x00000000;’ =>CIR 을 리셋 시킴. 모든 인터럽트를 비활성화한다.

Address offset: 0x0C Reset value: 0x0000 0000 Access: no wait state, word, half-word and byte access

-SetSysClock

```
/**
  * @brief  System Clock Source, PLL 곱셈기 & 나눗셈기, AHB/APBx Prescalers와 Flash 설정을 구성한다.
  * @Note   이 함수는 RCC Clock 구성을 Default Reset State로 Reset하기 위해 단 한 번만 호출된다.
  * @param  None
  * @retval None
  */
static void SetSysClock(void)
{
/******************************************************************************/
/*                PLL (clocked by HSE)을 System Clock Source로 사용한다.                */
/******************************************************************************/
  __IO uint32_t StartUpCounter = 0, HSEStatus = 0;

  /* HSE를 활성화 */
  RCC->CR |= ((uint32_t)RCC_CR_HSEON);

  /* Time Out되서 종료되거나 HSE가 종료될때까지 대기한다. */
  do
  {
    HSEStatus = RCC->CR & RCC_CR_HSERDY;                    HSE_STARTUP_TIMEOUT의 값은 0x05000
    StartUpCounter++;
  } while((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));

  if ((RCC->CR & RCC_CR_HSERDY) != RESET)          RESET = 0
  {
    HSEStatus = (uint32_t)0x01;
  }
  else
  {
    HSEStatus = (uint32_t)0x00;
  }
```

RCC_CR_HSEON 이라는 플래그를
활성화(bit16)

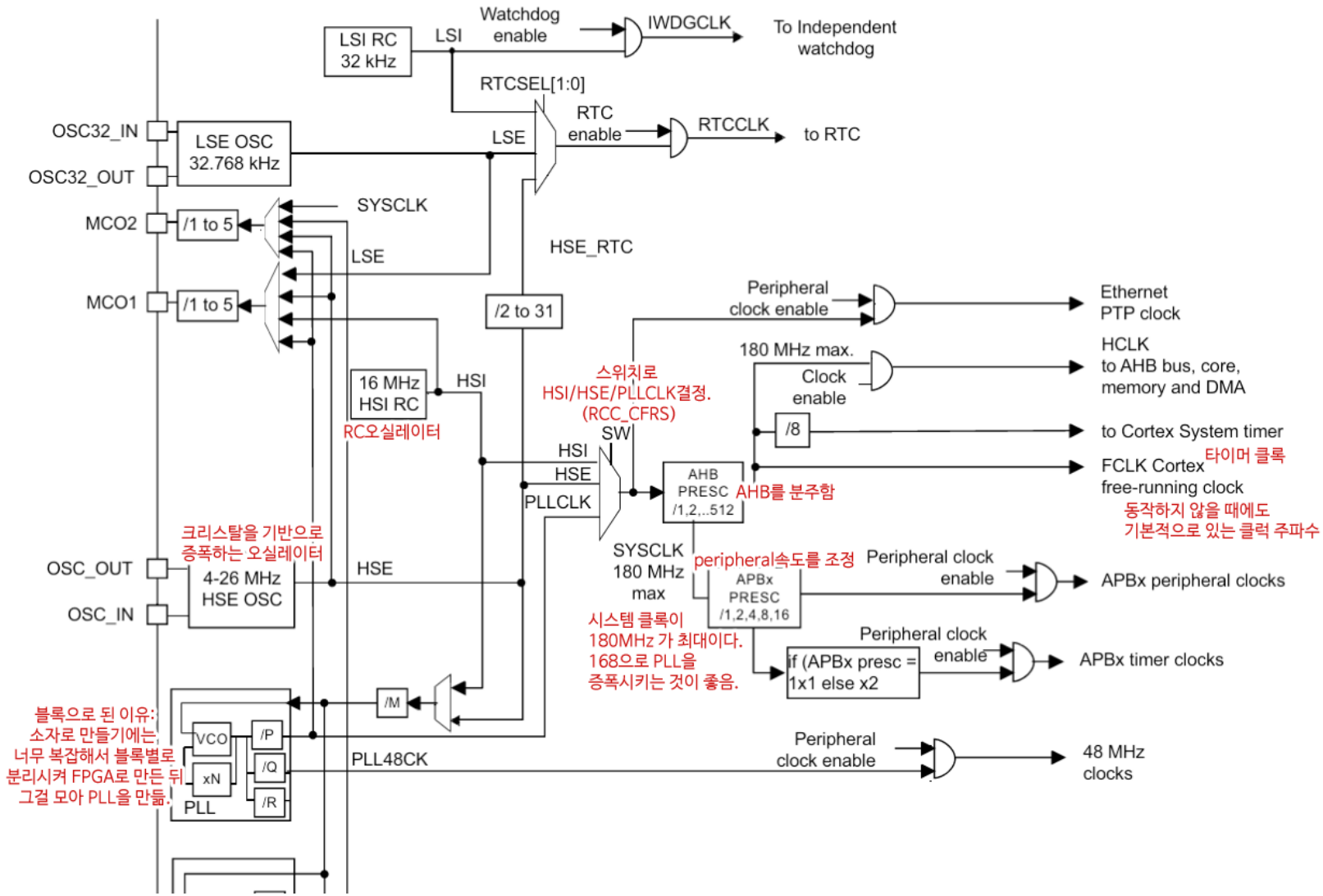
(CR bit 17) ready 되면 1, 안되어 있으면 0
Ready 가 되어있는지 계속 검사함
ready 는 하드웨어가 자동으로 세팅 해줌.
일정시간이 지나도 안정화가 안되면
오실레이터를 잘못 만들었다는 것을 알 수 있음
ready 가 1 이면 클럭 주파수가 안정화 되었다.

ready 가 되면 HSEStatus 에 1 을 넣고 되지
않으면 0 을 넣음.
0 일 때는 하드웨어 잘못 설계한 경우.

[RM0090 Reference Manual(STM32F407) - 6.3.1 RCC clock control register (RCC_CR)]

Bit 17
HSERDY: HSE clock ready flag Set by hardware to indicate that the HSE oscillator is stable. After the HSEON bit is cleared, HSERDY goes low after 6 HSE oscillator clock cycles.
0: HSE oscillator not read
1: HSE oscillator ready

-clock tree


LSI RC
32 kHz
LSI
Watchdog
enable
IWDGCLK
To Independent
watchdog
RTCSEL[1:0]
RTC
enable
RTCCLK
to RTC
OSC32_IN
OSC32_OUT
LSE OSC
32.768 kHz
LSE
MCO2
/1 to 5
SYSCLK
LSE
HSE_RTC
MCO1
/1 to 5
/2 to 31
Peripheral
clock enable
Ethernet
PTP clock
180 MHz max.
Clock
enable
HCLK
to AHB bus, core,
memory and DMA
16 MHz
HSI RC
HSI
RC오실레이터
스위치로
HSI/HSE/PLLCLK결정.
(RCC_CFRS)
/8
to Cortex System timer
SW
HSI
HSE
PLLCLK
AHB
PRESC
/1,2...512
AHB를 분주함
FCLK Cortex
free-running clock
타이머 클록
동작하지 않을 때에도
기본적으로 있는 클럭 주파수
크리스탈을 기반으로
증폭하는 오실레이터
OSC_OUT
OSC_IN
4-26 MHz
HSE OSC
HSE
SYSCLK
180 MHz
max
peripheral속도를 조정
APBx
PRESC
/1,2,4,8,16
Peripheral clock
enable
APBx peripheral clocks
시스템 클록이
180MHz 가 최대이다.
168으로 PLL을
증폭시키는 것이 좋음.
Peripheral clock
enable
if (APBx presc =
1x1 else x2
APBx timer clocks
/M
블록으로 된 이유:
소자로 만들기에는,
너무 복잡해서 블록별로
분리시켜 FPGA로 만든 뒤
그걸 모아 PLL을 만듦.
VCO
/P
/Q
/R
xN
PLL
PLL48CK
Peripheral
clock enable
48 MHz
clocks

```
  if (HSEStatus == (uint32_t)0x01)
  {
    /* High Performance Mode를 활성화하고, System Frequency를 168 MHz로 올린다. */
    RCC->APB1ENR |= RCC_APB1ENR_PWREN;
    PWR->CR |= PWR_CR_PMODE;
    RCC->APB1ENR |= RCC_APB1ENR_PWREN;

    /* HCLK = SYSCLK / 1*/
    RCC->CFGR |= RCC_CFGR_HPRE_DIV1;

    /* PCLK2 = HCLK / 2*/
    RCC->CFGR |= RCC_CFGR_PPRE2_DIV2;

    /* PCLK1 = HCLK / 4*/
    RCC->CFGR |= RCC_CFGR_PPRE1_DIV4;

    /* main PLL을 구성한다. */
    RCC->PLLCFGR = PLL_M | (PLL_N << 6) | (((PLL_P >> 1) -1) << 16) |
                   (RCC_PLLCFGR_PLLSRC_HSE) | (PLL_Q << 24);

    /* main PLL을 활성화 */
    RCC->CR |= RCC_CR_PLLON;

    /* main PLL이 준비될때까지 대기한다. */
    while((RCC->CR & RCC_CR_PLLRDY) == 0)
    {
    }

    /* Flash Prefetch, Instruction Cache, Data Cache를 구성하고 대기 상태 */
    FLASH->ACR = FLASH_ACR_ICEN |FLASH_ACR_DCEN |FLASH_ACR_LATENCY_5WS;

    /* System Clock Source로 main PLL을 선택한다. */
    RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
    RCC->CFGR |= RCC_CFGR_SW_PLL;

    /* System Clock Source로 main PLL이 사용될때까지 대기한다. */
    while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS ) != RCC_CFGR_SWS_PLL);
    {
    }
  }
  else
  { /* HSE가 Start-Up에 실패하면 Application은 잘못된 Clock을 구성할 것이다.
       사용자(학생분들)가 이러한 오류를 다루기 위한 Code를 이곳에 추가하면 된다. */
  }
}
```

RCC->APB1ENR |= RCC_APB1ENR_PWREN;

링크

[Cortex-M4 Technical Reference Manual]

http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf

[Cortex-M4 Device Generic User Guide]

http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf

[RM0090 Reference Manual(STM32F407)]

http://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf

[STM32F407 Datasheet]

http://www.st.com/content/ccc/resource/technical/document/datasheet/ef/92/76/6d/bb/c2/4f/f7/DM00037051.pdf/files/DM00037051.pdf/jcr:content/translations/en.DM00037051.pdf

http://www.jkelec.co.kr/img/arm/cortex-m4/stm32f4407igt6_evb/STM32F4x7-Datasheet.pdf

asm 으로 system call

```
#include<stdio.h>
#include<unistd.h>
#include<errno.h>
#include<stdlib.h>

int main(void)
{
	int i;
	unsigned int test_arr[7] = {0};

	register unsigned int *r0 asm("r0") = 0;
	register unsigned int r1 asm("r1") = 0;
	register unsigned int r2 asm("r2") = 0;
	register unsigned int r3 asm("r3") = 0;
	register unsigned int r4 asm("r4") = 0;
	register unsigned int r5 asm("r5") = 0;
	register unsigned int r6 asm("r6") = 0;
	register int r7 asm("r7") =  0;

	r0=test_arr;

	asm volatile("mov r1, #0x3₩n" //r1=0x3
			"mov r2, r1, lsl #2₩n" // r2=0b1100 = 12
			"mov r4, #0x2₩n" // r4 = 0x2
			"add r3, r1, r2, lsl r4₩n" // r3 = 48+3=51
			"stmia r0!, {r1,r2,r3}₩n" // r0 에 r1,r2,r3 를 넣고 위치 값 갱신
			"str r4, [r0]₩n" // r0 에 r4 레지스터 값을 넣음. 4 번째 배열에 2 가 들어감.
			"mov r5, #128₩n" // r5 = 128
			"mov r6, r5, lsr #3₩n" // r6 = 128/8 = 16
```

```
                "stmia r0, {r4, r5, r6}₩n" //r0 에 r4,r5,r6 넣기
                "sub r0, r0, #12₩n" // r0 의 위치 원래대로
                "ldmia r0, {r4,r5,r6}₩n"
                "swp r6,r3,[r0]");


    for(i=0; i<7; i++)
        printf("test_arr[%d] = %d₩n", i, test_arr[i]);
    printf("r4 = %u, r5 = %u, r6 = %u₩n", r4, r5, r6);

    r7 = 2; // r7 : system call 저장

    asm volatile("swi #0" : "=r" (r0) : "r" (r7) : "memory");

    if(r0 > 0) // parent
        printf("r0 = %p, Parent₩n", r0);
    else if(r0 == 0) // child
        printf("r0 = %p, Child₩n", r0);
    else //error
    {
        perror("fork()");
        exit(-1);
    }

    return 0;
}
```

```
xeno@xeno-NH:~/proj/0502$ qemu-arm-static -L /usr/arm-linux-gnueabi ./a.out
test_arr[0] = 51
test_arr[1] = 12
test_arr[2] = 51
test_arr[3] = 2
```

```
test_arr[4] = 128
test_arr[5] = 16
test_arr[6] = 0
r4 = 3, r5 = 12, r6 = 3
r0 = 0xe34, Parent
r0 = (nil), Child
```