2022년 6월 2일 - reset.S

- arch/arm/core/aarch32/cortex_m/reset.S
    - 리셋상태로 초기화한 뒤 C 초기화 코드로 분기하기 위한 준비
    - Privileged thread 모드 msp 에서 psp 로 전환
- armv8 mainline vs baseline
    - https://developer.arm.com/documentation/100688/0200/ARMv8-M-subprofiles
- ARMv8-M Architecture Technical Overview
    - https://community.arm.com/cfs-file/__key/telligent-evolution-components-attachments/01-2142-00-00-00-00-66-90/Whitepaper-_2D00_-ARMv8_2D00_M-Architecture-Technical-Overview.pdf
    - 한글: https://www.elec4.co.kr/article/articleView.asp?idx=17179
- https://en.wikipedia.org/wiki/ARM_Cortex-M
- NXP kinetis 같은 경우, 부팅 후 특정시간 이후에는 와치독 관련 설정이 불가능해지기 때문에 다른 초기화보다 선행되어야 함

2022년 6월 9일 - z_arm_prep_c()

- arch/arm/core/aarch32/prep_c.c
- VTOR
- XIP
- SW_VECTOR_RELAY, SW_VECTOR_RELAY_CLIENT

- FPU 접근 권한
    - CPACR 레지스터의 CP10 비트와 CP11 비트를 이용하여 FPU 접근 권한 설정 가능하다.
    - 스레드 실행 모드를 유저 모드로 설정한 경우 FP 레지스터 접근 권한을 Full access 로 설정하고, 아니라면 Privileged access 로 설정한다.
- Unshared FP 레지스터 모드
    - 하나의 스레드만 FPU를 사용하는 경우 FPCCR.ASPEN 비트(자동상태저장)와 FPCCR.LSPEN(레이지스태킹) 비트 모두 비 활성화한다.
- Shared FP 레지스터 모드
    - 2개 이상의 모든 스레드가 FPU 를 사용할 경우 FPCCR.ASPEN 비트(자동상태저장)와 FPCCR.LSPEN(레이지스태킹) 비트 모두 활성화한다.
- CONTROL.FPCA 비트
    - 이 비트는 FPU가 사용될 때 자동으로 설정되고, 문맥 전환 후 새로운 문맥이 시작되면 자동으로 지워진다.
- EXC_RETURN.FTYPE 비트
    - 이 비트는 스태킹되었던 스택 프레임의 형식(일반: 1, 확장 FP: 0)을 확인 가능하다.
    - 익셉션 리턴될 때 순차적으로 스태킹되었던 스택 프레임은 자동으로 언스태킹된다.

    - 스택 프레임의 형식에 따라 프레임 크기가 결정되므로 스택에서 인자를 추출할 경우 이를 고려하여 계산해야 한다.
    - 멀티 태스킹 환경에서 필요시 이 비트를 이용하여 FPU 사용 여부를 확인하여 FP 문맥을 저장/복원해야 한다.
- 스택 프레임 형식
    - 일반 : R0-R3, R12, LR, PC, xPSR
    - 확장 : 일반 스택 프레임 + FP (S0-S15, FPSCR)
- Lazy Stacking 활성화
    - Cortex-M4F 에서 지원이 되었다.
    - Lazy Stacking을 활성화하려면 FPCCR.ASPEN 비트(자동상태저장)와 FPCCR.LSPEN(레이지스태킹 활성화) 비트 모두 설정되어야 한다. (기본 값: 활성화)
    - 레이지 스태킹이 활성화된 경우 익셉션 엔트리시 FP 레지스터 스태킹을 하지 않는다.
    - FP 레지스터 스태킹할 공간만 예약하고 FPU를 사용하는 시점에 이전에 사용했던 FP 레지스터를 저장해온다.
- Lazy Stacking 비활성화
    - 항상 익셉션 엔트리시 FP 레지스터를 스태킹하고 익셉션 리턴시 언스태킹한다.
    - FP 레지스터 스태킹의 오버헤드로 인터럽트 대기 및 문맥 전환 시간이 증가될 수 있다.
- Lazy Stacking 참고
    - https://www.state-machine.com/doc/ARM-AN298.pdf

2022년 6월 16일

- ITCM/DTCM
    - https://www.st.com/resource/en/application_note/an4667-stm32f7-series-system-architecture-and-performance-stmicroelectronics.pdf
- https://docs.zephyrproject.org/latest/build/dts/dt-vs-kconfig.html

깃허브 ID

- @onkwon
- @ys-oh
- @kyehyun
- @jobaek
- @kkoroke
- @cjujh85
- @bioman-3d
- @inhyunhwang

2022년 6월 20일

- SCB_SCR_SERVONPEND => wakeup event 초기화
    - https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-processor/power-management/wakeup-from-sleep-mode
- mpu_init()
    - Inner / outter memory attribute setting

- https://github.com/zephyrproject-rtos/zephyr/pull/31481
- https://docs.zephyrproject.org/latest/kernel/memory_management/slabs.html
- https://developer.arm.com/documentation/ecm0359818/latest/
- https://docs.zephyrproject.org/latest/services/logging/index.html

2022년 7월 4일

- kernel/igtnit.c line423 시작
- Stack guard 초기화
    - Random stack guard for Stack canaries
    - RNG (Random number generator) support
        - Entropy driver
        - drivers/entropy/entropy_stm32.c -> entropy_stm32_rng_get_entrpy()
        - https://www.st.com/content/ccc/resource/training/technical/product_training/group0/5a/ba/43/28/46/d0/44/aa/STM32H7-Security-Random_Number_Generator_RNG/files/STM32H7-Security-Random_Number_Generator_RNG.pdf/_jcr_content/translations/en.STM32H7-Security-Random_Number_Generator_RNG.pdf
        - If Insufficient performance -> XOROSHIRO algorithm
            - https://prng.di.unimi.it/xoroshiro128plus.c
    - TEST random generator
        - Not truly random number generator
        - Should not be used in a production. Only Test
- Timing 초기화
    - Board level
        - nothing (AT TAG V3.0.0)
    - SoC level
        - soc/arm/nordifc_nrf/timing.c
    - Arch level
        - DWT(Debug watchpoint trace) - Cycle counter
        - Timing on code execution
        - https://developer.arm.com/documentation/100734/0100/Debug-for-ARMv8-M/Other-debug-functionality/DWT
    - Atomic functions (GCC compiler built-in functions)

        - __atomic_load_n : read
        - __atomic_fetch_add : add
        - __atomic_fetch_sub : sub
        - __atomic_compare_exchange_n : compare and exchange
        - Compare and swap(CAS): https://en.wikipedia.org/wiki/Compare-and-swap
    - Timing functions may be used for a benchmark,
- Ready queue 초기화
    - kernel/init.c prepare_multithreading() - z_sched_init()
    - SMP One CPU pinning option
        - symmetric multiprocessing
        - Every thread is pined to one CPU
    - ready_q
        - SMP - _kernel.cpus[].ready_q
        - Non SMP - _kernel.ready_q
    - Implement
        - rb tree (Red/black tree)
            - Scalable but, Overhead ~2 kb extra code size
            - Runnable more than 20 threads roughly
            - During insert new rb node, compare priority of new rb node and existing rb node using lessthan function
                - .lessthan_fn = z_priq_rb_lessthan()
        - Multi queue
            - Classic algorithm and tiny code size
            - O(1) time but, large RAM budget
            - Runnable small numbers of threads
    - SMP vs AMP
        - https://elinux.org/images/e/e5/Multi-core_application_development_with_Zephyr_RTOS_-_2019.10.23.pdf
- kernel/sched.c line1160 끝

2022년 7월 11일

- kernel/init.c line437 시작 TIMESLICING
- kernel/Kconfig TIMESLICING , TIMESLICE_SIZE,TIMESLICE_PRIORITY
- time slice set
    - Sched.c line 408
    - Time_units.h line 388 k_ms_to_ticks_ceil64(uint64_t t)
    - Time_units.h line 101 z_tmcvt(uint64 t, uintt32_t from_hz)
    - Sched.c line 396 z_reset_time_slice(void)
- kernel/sched.c line 422 끝

2022년 7월 25일

- **kernel/init.c line439** 시작
  ARCH_SWITCH_TO_MAIN_NO_MULTITHREADING
- **aarch32/thread.c line586**
  #if !defined(CONFIG_MULTITHREADING) && defined(CONFIG_CPU_CORTEX_M)
  *FUNC_NORETURN z_arm_switch_to_main_no_multithreading (k_thread_entry_t main_entry, void *p1, void *p2, void *p3)*
  Function Summary:
  PSPLIM 및 PSP 값 설정 및 main_entry(=bg_thread_main)으로 branch
  Commit :
  멀티스레딩이 지원되지 않는(CONFIG_MULTITHREADING=n) Zephyr를 빌드하는 경우 cstart()에서 main()으로 전환하는 사용자 지정(ARCH 관련) 함수를 소개합니다. 이것은 Cortex-M 초기화 코드가 일시적으로 인터럽트 스택을 사용하고 main()이 대신 z_main_stack을 사용해야 하기 때문에 필요합니다. 이 함수는 PSP 전환, PSPLIM 설정(ARMv8-M용), FPU 초기화 및 정적 메모리 영역 초기화를 수행하여 일반(CONFIG_MULTITHREADING=y) 경우를 모방합니다.
  - **aarch32/thread.c line591**
    void *z_arm_prepare_switch_to_main(void)*
    Function Summary:
    FPSCR (Floating-Point Status and Control Register) &
    FPCA(Floating-Point Context Address Register) 초기화
    Comment:
    다중 스레딩 지원 여부에 관계없이 Zephyr를 실행하는 경우에 적용 가능한 Cortex-M 초기화를 위한 내부 기능입니다. 비공유 FP 레지스터 모드에서 부동 소수점 상태 및 제어 레지스터 초기화(공유 FP 레지스터 모드에서 FPSCR은 FP를 사용하는 스레드에 대한 스레드 생성 시 초기화됨)
- **kernel/init.c line164**
  void *bg_thread_main(void *unused1, void *unused2, void *unused3)*
  Function Summary:
  TBD
  Comment:
  이 루틴은 나머지 init 함수를 호출하여 커널 초기화를 완료한 다음 애플리케이션의 main() 루틴을 호출합니다. 백업 저장 또는 축출 알고리즘이 커널 개체를 초기화하고 모든 POST_KERNEL 및 이후 작업이 메모리 관리 작업을 수행할 수 있도록 여기에서 호출됩니다. (언제든지 허용되는 z_phys_map() 제외).
- **kernel/mmu.c line835**
  void *z_mem_manage_init(void)*
  Function Summary:
  MMU 초기화 (TBD)
  Commit:
  부팅 시 페이지 프레임 ontology를 초기화하고 메모리 매핑을 수행할 때 업데이트합니다.
  - **kernel/mmu.c line842**

void *free_page_frame_list_init(void)*
Free_page_frame_list (Singled Linked List) 초기화

- **kernel/include/mmu.h line50**
  #define Z_BOOT_VIRT_TO_PHYS(virt)
  인자로 받은 가상주소를 물리주소로 변환
- **kernel/include/mmu.h line51**
  #define Z_BOOT_PHYS_TO_VIRT(phys)
  인자로 받은 물리주소를 가상주소로 변환
- **kernel/mmu.c line185**
  *struct z_page_frame *z_phys_to_page_frame(uintptr_t phys)*
  물리주소 값을 인자로 물리적인 RAM의 page_frames 주소값을 return

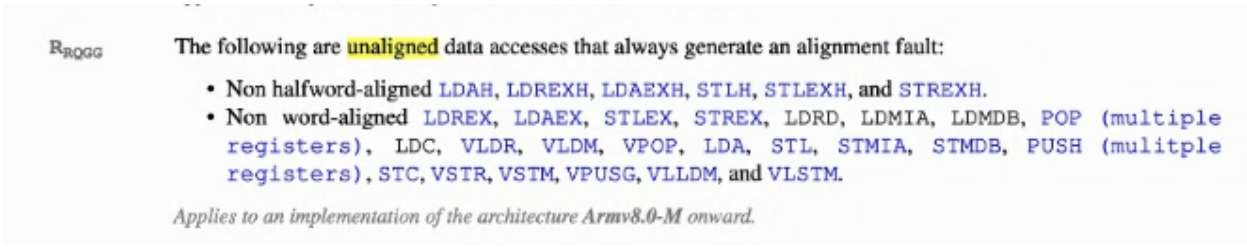
- kernel/mmu.c line 860 끝 (line 872 할 차례)

2022년 8월 22일

- Ctor, init_array
  - https://maskray.me/blog/2021-11-07-init-ctors-init-array
  - https://developer.arm.com/documentation/dui0475/c/the-arm-c-and-c---libraries/c--initialization--construction-and-destruction
  - https://developer.arm.com/documentation/dui0475/c/the-arm-c-and-c---libraries/legacy-support-for-c--pi-ctorvec-instead-of--init-array
  -

2022년 9월 5일 - async

- drivers/spi/spi_ll_stm32.c
  - UNALIGNED_GET()
    - Prevent Unaligned access (depends on Architecture)
  -



R_RQGG The following are unaligned data accesses that always generate an alignment fault:

- Non halfword-aligned LDAH, LDREXH, LDAEXH, STLH, STLEXH, and STREXH.
- Non word-aligned LDREX, LDAEX, STLEX, STREX, LDRD, LDMIA, LDMDB, POP (multiple registers), LDC, VLDR, VLDM, VPOP, LDA, STL, STMIA, STMDB, PUSH (mulitple registers), STC, VSTR, VSTM, VPUSG, VLLDM, and VLSTM.

*Applies to an implementation of the architecture **Armv8.0-M** onward.*

2022년 9월 12일

2022년 9월 19일

- USB 관련 문서
  - https://www.keil.com/pack/doc/mw/USB/html/index.html
  - https://www.beyondlogic.org/usbnutshell/usb1.shtml
  - https://www.infineon.com/dgdl/Infineon-AN57294_USB_101_An_Introduction_to_Universal_Serial_Bus_2.0-ApplicationNotes-v09_00-EN.pdf?fileId=8ac78c8c7cdc391c017d072d8e8e5256
  - https://doc.lagout.org/science/0_Computer%20Science/9_Others/9_Misc/USB%20Complete%20The%20Developer%27s%20Guide%204th%20Ed.pdf
  - https://www.usb.org/defined-class-codes
- USB 2.0 Specifications