에디로봇아카데미
임베디드 마스터 Lv1 과정

제 5기

2023. 06. 15

박 상호

C Inline Assembly (ARM 32-Bit)

EDDI
Electronic Design Development Institute

1. Inline Assembly

인라인 어셈블러는 C 코드에 어셈블리 명령어를 삽입하는 것을 의미하며 추가 어셈블리 및 링크 단계를 수행하지 않고 C 코드에 Assembly 코드를 포함할 수 있다.

2. 장단점

- C 언어 변수, 함수와 혼합하여 사용 가능하며 이를 통해 C로 구현 시 복잡한 동작을 어셈블리 명령으로 단순하게 처리하여 최적화 할 수 있다.
- 컴파일러에서 생성하는 함수 프롤로그나 에필로그를 생략할 수 있고, 오버헤드를 최소화 할 수 있다.
- Inline Assembly 사용 시 Architecture 가 다른 System에서는 해당 Code를 사용하기 힘들어 범용성이 떨어진다.

3. 사용법

- Register Keyword (Register Variable)

register (Variable Type) (Variable Name) asm (“(Register Name)”) = (Value);

- 특정 변수가 Memory 가 아닌 Register 공간을 사용하도록 지정한다. 이를 통해 Memory 접근 시간을 최소화하여 처리 속도를 향상시킬 수 있다.
- Memory 공간을 사용하는 것이 아니기에 변수 주소를 구할 수 없고, 이로 인해 배열 구성 시 Pointer로 접근이 불가능하다.
- Register는 매우 한정된 자원이므로 무한히 생성할 수 없으며, 전역 변수로 사용할 수 없다.

- Inline Assembly

asm volatile (“(Assembly Code)” :(Output) :(Input) :(Clobber));

- volatile : 컴파일러가 해당 Code를 최적화 하지 않고, 사용자가 입력한 코드를 그대로 수행하도록 한다.
- :(Output Operand) : 코드 수행 결과가 저장 되는 출력 변수
- :(Input Operand) : 코드 수행에 필요한 입력 변수
- :(Clobber) : 임의의 Register가 Inline Assembly Code 에서 사용 될 경우, 해당 Register가 In/Output Operand로 사용되지 않도록 설정한다.

C Inline Assembly (ARM 32-Bit)

EDDI
Electronic Design Development Institute

4. In/Output Operand

- Inline Assembly Code 사용 시 코드 수행에 필요한 입력, 코드 수행 결과가 저장 되는 출력 변수를 지정하는 인자

Operand		Modifier	
m	Memory 인자	=	출력 인자 임을 나타냄
r	General Purpose Register 인자	+	입/출력 모두 가능한 인자
i	상수 (심볼릭 상수 포함)	&	출력 인자에 사용하는 Early Clobber로 컴파일러가 출력 인자와 입력인자를 서로 다른 Register에 할당하도록 한다.
n	즉시 값 (정수 피연산자)		
F	부동소수점 상수		
g	특수 목적 Register를 제외한 GPR		

C Inline Assembly (Non-Inline ASM)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function can_I_execute_asm:
=> 0x00010474 <+0>:     push    {r11, lr}
   0x00010478 <+4>:     add     r11, sp, #4
   0x0001047c <+8>:     sub     sp, sp, #24
   0x00010480 <+12>:    mov     r3, #0
   0x00010484 <+16>:    str     r3, [r11, #-24] ; 0xffffffe8
   0x00010488 <+20>:    mov     r3, #0
   0x0001048c <+24>:    str     r3, [r11, #-20] ; 0xffffffec
   0x00010490 <+28>:    mov     r3, #0
   0x00010494 <+32>:    str     r3, [r11, #-16]
   0x00010498 <+36>:    mov     r3, #0
   0x0001049c <+40>:    str     r3, [r11, #-12]
   0x000104a0 <+44>:    mov     r3, #0
   0x000104a4 <+48>:    str     r3, [r11, #-8]
   0x000104a8 <+52>:    ldr     r3, [pc, #116]  ; 0x10524 <can_I_execute_asm+176>
   0x000104ac <+56>:    str     r3, [r11, #-8]
   0x000104b0 <+60>:    mov     r3, #3
   0x000104b4 <+64>:    str     r3, [r11, #-24] ; 0xffffffe8
   0x000104b8 <+68>:    ldr     r3, [r11, #-24] ; 0xffffffe8
   0x000104bc <+72>:    lsl     r3, r3, #2
   0x000104c0 <+76>:    str     r3, [r11, #-20] ; 0xffffffec
   0x000104c4 <+80>:    mov     r3, #2
   0x000104c8 <+84>:    str     r3, [r11, #-12]
   0x000104cc <+88>:    ldr     r2, [r11, #-20] ; 0xffffffec
   0x000104d0 <+92>:    ldr     r3, [r11, #-12]
   0x000104d4 <+96>:    lsl     r3, r2, r3
   0x000104d8 <+100>:   ldr     r2, [r11, #-24] ; 0xffffffe8
   0x000104dc <+104>:   add     r3, r2, r3
   0x000104e0 <+108>:   str     r3, [r11, #-16]
   0x000104e4 <+112>:   ldr     r3, [r11, #-8]
   0x000104e8 <+116>:   ldr     r2, [r11, #-24] ; 0xffffffe8
   0x000104ec <+120>:   str     r2, [r3]
   0x000104f0 <+124>:   ldr     r3, [r11, #-8]
   0x000104f4 <+128>:   add     r3, r3, #4
   0x000104f8 <+132>:   ldr     r2, [r11, #-20] ; 0xffffffec
   0x000104fc <+136>:   str     r2, [r3]
   0x00010500 <+140>:   ldr     r3, [r11, #-8]
   0x00010504 <+144>:   add     r3, r3, #8
   0x00010508 <+148>:   ldr     r2, [r11, #-16]
   0x0001050c <+152>:   str     r2, [r3]
   0x00010510 <+156>:   ldr     r0, [pc, #12]   ; 0x10524 <can_I_execute_asm+176>
   0x00010514 <+160>:   bl      0x10400 <print_arr_info>
   0x00010518 <+164>:   nop                     ; (mov r0, r0)
--Type <RET> for more, q to quit, c to continue without paging--
   0x0001051c <+168>:   sub     sp, r11, #4
   0x00010520 <+172>:   pop     {r11, pc}
   0x00010524 <+176>:   andeq   r1, r2, r8, lsr #32
End of assembler dump.
```

```
Dump of assembler code for function yes_you_can_execute_asm:
=> 0x00010528 <+0>:     push    {r11, lr}
   0x0001052c <+4>:     add     r11, sp, #4
   0x00010530 <+8>:     sub     sp, sp, #8
   0x00010534 <+12>:    mov     r3, #0
   0x00010538 <+16>:    str     r3, [r11, #-12]
   0x0001053c <+20>:    mov     r3, #0
   0x00010540 <+24>:    str     r3, [r11, #-8]
   0x00010544 <+28>:    ldr     r3, [pc, #36]   ; 0x10570 <yes_you_can_execute_asm+72>
   0x00010548 <+32>:    str     r3, [r11, #-8]
   0x0001054c <+36>:    ldr     r3, [r11, #-8]
   0x00010550 <+40>:    ldr     r3, [r3, #8]
   0x00010554 <+44>:    str     r3, [r11, #-12]
   0x00010558 <+48>:    ldr     r1, [r11, #-12]
   0x0001055c <+52>:    ldr     r0, [pc, #16]   ; 0x10574 <yes_you_can_execute_asm+76>
   0x00010560 <+56>:    bl      0x102e0 <printf@plt>
   0x00010564 <+60>:    nop                     ; (mov r0, r0)
   0x00010568 <+64>:    sub     sp, r11, #4
   0x0001056c <+68>:    pop     {r11, pc}
   0x00010570 <+72>:    andeq   r1, r2, r8, lsr #32
   0x00010574 <+76>:    andeq   r0, r1, r4, lsl r6
End of assembler dump.
```

```
/usr/arm-linux-gnueabi non_inlined_asm
arr[0] = 3
arr[1] = 12
arr[2] = 51
arr[3] = 4
arr[4] = 5
arr[5] = 0
qemu-arm-static: QEMU: Terminated via GDBstub
```

C Inline Assembly (Use Inline ASM)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function can_I_execute_asm:
=> 0x00010474 <+0>:     push    {r4, r5, r11, lr}
   0x00010478 <+4>:     add     r11, sp, #12
   0x0001047c <+8>:     mov     r1, #0
   0x00010480 <+12>:    mov     r2, #0
   0x00010484 <+16>:    mov     r3, #0
   0x00010488 <+20>:    mov     r4, #0
   0x0001048c <+24>:    mov     r5, #0
   0x00010490 <+28>:    ldr     r5, [pc, #32]   ; 0x104b8 <can_I_execute_asm+68>
   0x00010494 <+32>:    mov     r1, #3
   0x00010498 <+36>:    lsl     r2, r1, #2
   0x0001049c <+40>:    mov     r4, #2
   0x000104a0 <+44>:    add     r3, r1, r2, lsl r4
   0x000104a4 <+48>:    stm     r5, {r1, r2, r3}
   0x000104a8 <+52>:    ldr     r0, [pc, #8]    ; 0x104b8 <can_I_execute_asm+68>
   0x000104ac <+56>:    bl      0x10400 <print_arr_info>
   0x000104b0 <+60>:    nop                     ; (mov r0, r0)
   0x000104b4 <+64>:    pop     {r4, r5, r11, pc}
   0x000104b8 <+68>:    andeq   r1, r2, r8, lsr #32
End of assembler dump.
   0x000104c0 <+76>:    str     r3, [r11, #-20] ; 0xffffffec
   0x000104c4 <+80>:    mov     r3, #2
   0x000104c8 <+84>:    str     r3, [r11, #-12]
   0x000104cc <+88>:    ldr     r2, [r11, #-20] ; 0xffffffec
   0x000104d0 <+92>:    ldr     r3, [r11, #-12]
   0x000104d4 <+96>:    lsl     r3, r2, r3
   0x000104d8 <+100>:   ldr     r2, [r11, #-24] ; 0xffffffe8
   0x000104dc <+104>:   add     r3, r2, r3
   0x000104e0 <+108>:   str     r3, [r11, #-16]
   0x000104e4 <+112>:   ldr     r3, [r11, #-8]
   0x000104e8 <+116>:   ldr     r2, [r11, #-24] ; 0xffffffe8
   0x000104ec <+120>:   str     r2, [r3]
   0x000104f0 <+124>:   ldr     r3, [r11, #-8]
   0x000104f4 <+128>:   add     r3, r3, #4
   0x000104f8 <+132>:   ldr     r2, [r11, #-20] ; 0xffffffec
   0x000104fc <+136>:   str     r2, [r3]
   0x00010500 <+140>:   ldr     r3, [r11, #-8]
   0x00010504 <+144>:   add     r3, r3, #8
   0x00010508 <+148>:   ldr     r2, [r11, #-16]
   0x0001050c <+152>:   str     r2, [r3]
   0x00010510 <+156>:   ldr     r0, [pc, #12]   ; 0x10524 <can_I_execute_asm+176>
   0x00010514 <+160>:   bl      0x10400 <print_arr_info>
   0x00010518 <+164>:   nop                     ; (mov r0, r0)
--Type <RET> for more, q to quit, c to continue without paging--
   0x0001051c <+168>:   sub     sp, r11, #4
   0x00010520 <+172>:   pop     {r11, pc}
   0x00010524 <+176>:   andeq   r1, r2, r8, lsr #32
End of assembler dump.
```

```
Dump of assembler code for function yes_you_can_execute_asm:
=> 0x000104bc <+0>:     push    {r11, lr}
   0x000104c0 <+4>:     add     r11, sp, #4
   0x000104c4 <+8>:     mov     r0, #0
   0x000104c8 <+12>:    mov     r1, #0
   0x000104cc <+16>:    ldr     r1, [pc, #28]   ; 0x104f0 <yes_you_can_execute_asm+52>
   0x000104d0 <+20>:    mov     r2, #8
   0x000104d4 <+24>:    ldr     r0, [r1, r2]
   0x000104d8 <+28>:    mov     r3, r0
   0x000104dc <+32>:    mov     r1, r3
   0x000104e0 <+36>:    ldr     r0, [pc, #12]   ; 0x104f4 <yes_you_can_execute_asm+56>
   0x000104e4 <+40>:    bl      0x102e0 <printf@plt>
   0x000104e8 <+44>:    nop                     ; (mov r0, r0)
   0x000104ec <+48>:    pop     {r11, pc}
   0x000104f0 <+52>:    andeq   r1, r2, r8, lsr #32
   0x000104f4 <+56>:    muleq   r1, r4, r5
End of assembler dump.
   0x00010568 <+64>:    sub     sp, r11, #4
   0x0001056c <+68>:    pop     {r11, pc}
   0x00010570 <+72>:    andeq   r1, r2, r8, lsr #32
   0x00010574 <+76>:    andeq   r0, r1, r4, lsl r6
End of assembler dump.
```

```
/usr/arm-linux-gnueabi inlined_asm
arr[0] = 3
arr[1] = 12
arr[2] = 51
arr[3] = 4
arr[4] = 5
arr[5] = 0
qemu-arm-static: QEMU: Terminated via GDBstub
```

Inline Assembly를 사용함으로써 기존 C 코드 대비

Assembly Code Size 가 크게 감소되었음을 확인 가능

Agile 방법론

EDDI
Electronic Design
Development Institute

1. Agile 방법론이란?

애자일(Agile)의 사전적 의미는 '날렵한, 민첩한'을 의미하며, 장기 계획에 의존적이고 단계 별로 진행되는 '폭포수 모델' 과 달리 단기 계획을 갖고 주기적으로 검토하며 필요할 때마다 요구사항을 더하고 수정하여 유연성과 적응성을 추구하는 프로그래밍 기법

2. Agile 방법론의 핵심 가치 (애자일 개발 선언문)

- 공정과 도구보다 개인과 상호작용을
- 포괄적인 문서보다 작동하는 소프트웨어를
- 계약 협상보다 고객과의 협력을
- 계획을 따르기 보다 변화에 대응하기를

3. Agile 방법론의 장점

- 변화되는 요구 사항을 수용하기 쉽고, 최종 목표가 확실하지 않은 프로젝트에 용이하고 고객 요구 사항에 대응하기 쉽다.
- 즉각적인 피드백을 통해 빠른 프로토타입 제작이 가능하다.
- 프로젝트 단위를 세분화하면 높은 품질의 빠른 개발, 테스트가 가능하며, 프로젝트 기간을 단축시킬 수 있다.
- 팀 상호 작용을 통해 팀워크가 향상된다.

4. Agile 방법론의 단점

- 상위 관리자 중점의 수직 서열 조직 문화와는 잘 맞지 않다.
- 스프린트에 대한 경험이 있으면서 빠른 반복 작업에 익숙한 스크럼 마스터가 필요하다.
- 팀원이 잘 조직되지 않거나 방법론에 대한 이해도가 떨어지는 경우, 자립성이 없는 경우, 애자일 방법론을 채택하면 문제가 발생할 수 있다.

5. Agile의 주요 요소

- Scrum
- Xp(eXtreme Programing)

Scrum

EDDI
Electronic Design
Development Institute

1. Scrum 이란?

상호, 점진적 개발 방법론으로, 고객 요구사항 충족에 초점을 맞추고, 목표를 짧은 주기로 점진적이며 경험적으로 시스템을 지속해서 개발하는 기법

2. Scrum 의 핵심 가치

- 용기 : 고객의 요구 사항이나 기능이 이해 되지 않거나 문제가 있다면 말할 수 있어야 하고, 문제에 도전적으로 시도해보는 용기와 완료 할 수 없는 업무량 또는 어떠한 문제로 업무를 완료할 수 없다면 말 할 수 있어야 한다.
- 집중 : 모든 노력과 기술은 현재 프로젝트의 성공을 위해 집중하고, 업무에 집중 할 수 있도록 불필요한 회의 참석은 지양하며, 단순 반복 작업은 제거 하거나 자동화해야 한다.
- 약속 (헌신/책임) : 팀의 목표 달성을 위해 개개인이 공약한 목표 달성을 위해 팀에 헌신하며, 이를 달성 위해 필요한 최대한 지원과 권한이 필요하다.
- 존중 : 팀의 협업을 지원하기 위해 팀 멤버들은 다른 멤버, 스크럼 마스터, 스크럼 프로세스를 존중해야 한다.
- 투명성/개방성 : 프로젝트에 대한 백로그, 스크럼 회의, 스프린트 리뷰를 통해 자신에게 불리해도 모든 문제를 공유하고 숨기지 않고 도움을 요청한다.

3. Scrum 역할

- Project Owner : 제품의 책임자이며, 제품에 대한 고객 요구 사항을 정의하고 백로그를 주로 작성한다.
- Scrum Master : 스크럼이 잘 진행될 수 있도록 돕는다. 팀원들이 스크럼 중 이슈를 겪을 경우, 이를 해결하기 위한 중개자 역할을 하며, 일일 미팅을 주도하고 스프린트 플래닝, 리뷰를 주최하고, 외부 팀과 협업이 필요할 경우 소통 채널의 역할을 수행한다.
- Developer : 프로그래머, QA, 기획자, 디자이너등, 제품의 요구사항을 구현하기 위해 기여할 수 있는 모든 사람을 칭한다.

4. Scrum 진행 방식

- 스프린트, 제품 백로그를 작성한다. 스크럼 스프린트를 시작하려면 먼저 팀 리더(스크럼 마스터)가 수행할 업무를 파악하고, 명확하게 문서화해야 한다.
- 스프린트 플래닝 세션을 수행하여 팀이 특정 스프린트 동안 백로그의 어떤 업무에 주력할 것인지 우선 순위를 평가한다.
- 업무를 작은 단위로 나누어 스크럼 스프린트를 수행한다. 스프린트는 주 단위 동안 진행되며, 팀은 스프린트 플래닝 세션에서 정리한 우선 순위에 따라 백로그 항목을 수행한다.
- 매일 15분 동안 모든 팀원이 미팅에 참가하여 현재 진행 중인 업무에 관해 자세히 보고하고, 예상치 못한 문제에 부딪히면 이를 논의하여 우선 순위를 조정한다.
- 스프린트 리뷰에서 업무 내용을 공유하고 반영합니다. 스프린트가 끝난 후 스프린트 전체를 되돌아보고 업무 어떻게 진행 되었는 지, 진행 과정에서의 문제점과 반성, 앞으로의 개선점을 논의한다.

Xp (eXtreme Programing)

EDDI
Electronic Design Development Institute

1. Xp 란?

고객의 요구 사항을 한번에 정의하는 방식이 아닌 변화하는 고객의 요구사항에 명확하게 대응 하기 위해 고객이 직접 참여, 확인하고, 반복형 모델의 개발 주기를 극단적으로 짧게 하여 프로그래머가 설계, 구현, 시험 활동을 전체 SW 개발 기간에 걸쳐 조금씩 자주 시행하도록 하는 기법이다.

2. Xp의 기본 원리

- Pair programming : 개발자 둘이서 짝으로 코딩한다.
- Collective Ownership (공동 코드 소유) : 시스템에 있는 코드는 누구든지 언제라도 수정 가능하다.
- CI (Continuous Integration, 지속적인 통합) : 작업이 끝날 때 마다 지속하여 여러 번 소프트웨어를 통합하고 빌드할 수 있다.
- Planning Process (계획 세우기) : 고객 요구 사항을 정의하고, 개발자가 필요한 것은 무엇이며 어떤 부분에서 지연될 수 있는지를 알려주어야 한다.
- Small Release (작은 릴리즈) : 작은 시스템을 만들어 릴리즈 기간을 짧은 단위로 반복함으로써 고객의 요구 변화에 신속히 대응한다.
- Metaphor (메타포어) : 공통적인 문장 형태로 작성한 시스템 아키텍처 기술과 이름 체계를 통해 고객과 개발자 간의 의사소통을 원활하도록 한다.
- Simple Design (간단한 디자인) : 가능한한 단순하게 요구 사항을 충족하는 시스템을 구현, 설계한다.
- TDD (Test Driven Development, 테스트 기반 개발) : 기능 구현에 앞서 테스트를 먼저 수행하고, 이 테스트를 통과할 수 있도록 코드를 작성한다.
- Refactoring (리팩토링) : 프로그램의 기능을 바꾸지 않으면서 중복 제거, 단순화 등을 위해 시스템을 재구성한다. (코드를 수정한다.)
- On-Site Customer(고객 상주) : 개발자들의 질문에 즉각 대답해 줄 수 있는 고객을 프로젝트에 풀타임으로 상주시킨다.
- Coding Standard (코드 표준) : 효과적인 공동 작업을 위해 모든 코드에 대한 코딩 표준을 관례에 따라 정의한다.

3. Xp 개발 프로세스

- 사용자 스토리 : 고객 요구 사항을 간단한 시나리오로 표현하고 내용은 기능 단위로 구성하며, 필요한 경우 간단한 테스트 사항도 기재한다. 사용자 스토리는 사용자의 입장에서 작성 되야 하며, 개발자 입장에서 재해석해서 적으면 안된다.
- 스파이크 : 요구 사항의 신뢰성을 높이고 기술 문제에 대한 위험을 감소시키기 위해 프로젝트 시작전에 핵심기능을 별도로 만드는 간단한 프로그램
- 릴리즈 계획 수립 : 프로젝트의 부분 혹은 전체에 대한 배포 일정을 수립한다.
- 인수 테스트 : 계획된 릴리즈가 구현되면 사용자 스토리 작성 시 기재한 테스트 시나리오를 사용자가 직접 수행한다. 테스트 이후 새로운 요구사항이 작성되거나 요구사항의 상대적 우선순위가 변경될 수 있다. 인수 테스트를 통과해야 스토리 처리가 완료되며 테스트를 통과하면 소규모 릴리즈를 실시한다.
- 소규모 릴리즈 : 릴리즈를 소규모로 하게 되면, 고객의 반응을 기능별로 확인할 수 있어, 고객의 요구 사항 변화에 좀 더 유연하게 대응할 수 있다. 현재 릴리즈가 완제품이 아닌 경우 다음 릴리즈 일정에 맞게 개발 진행한다.

DDD (Domain Driven Development)

EDDI
Electronic Design
Development Institute

1. DDD (Domain Driven Development) 란?

특정 기능이나 작업을 수행하는 모듈을 따로 설계하고 구현하는 기능 모듈 구현과는 달리 도메인이라 칭하는 유사한 기능의 집합으로, 즉 비즈니스 로직과 문제를 중심으로 소프트웨어를 설계하고 구현하는 기법

--- 향 후 내용 이해 후 다시 정리...

DDD Example Code (main/)

EDDI
Electronic Design Development Institute

1. main.c

```c
#include <stdio.h>

#include "add_on_lib.h"
#include "../receiver/receiver.h"

int main (void)
{
    // F7 : Build
    // Shift + F5 : Run;
    // Ctrl + F5 : Debug

    printf("First Test\n");
    iam_add_on();

    recv_command_from_outbound();

    return 0;
}
```

main 함수는 Top Design 으로

1. printf 함수를 사용하여 “First Test₩n” 을 출력

2. iam_add_on 함수 실행
   - printf 함수를 사용하여 “I’am add on₩n”을 출력

3. recv_command_from_outbound 함수 실행
   - printf 함수를 사용하여 "외부에서 명령이 들어온다 가정 (랜덤 생성)₩n” 출력
   - rand = srand(time(NULL)) : rand 에 seed 값 생성 후 대입
   - command = rand % 11 : command 에 0 ~ 10 사이의 랜덤 값 생성 후 대입
   - printf 함수를 사용하여 "수신된 명령 : %d₩n“ command 값을 출력
   - protocol_call_table[command] 함수 포인터 배열을 통해 command 값에 따라 Handler Method 실행

4. return 0; (Stack Frame 해제 및 복귀 주소 설정)

을 수행한다.

DDD Example Code (main/)

EDDI
Electronic Design Development Institute

2. add_on_lib.h

```
#ifndef __ADD_ON_LIB_H__
#define __ADD_ON_LIB_H__

void iam_add_on (void);

#endif
```

3. add_on_lib.c

```
#include "add_on_lib.h"
#include <stdio.h>

void iam_add_on (void)
{
    printf("I\'m add on\n");
}
```

add_on_lib Library는

iam_add_on 함수를 포함하는 Library로

iam_add_on 함수는 실행 시

printf 함수를 사용하여 "I'am add on₩n"을 출력한다.

add_on_lib.h 시작 부분은

#ifndef __ADD_ON_LIB_H__
#define __ADD_ON_LIB_H__

...
#endif

는 컴파일 시 Header 파일의 순환 참조나 중복 포함을

방지하기 위한 전 처리 구문으로 include guards 라 불린다.

DDD Example Code (protocol/)

EDDI
Electronic Design Development Institute

1. protocol.h

```
#ifndef __PROTOCOL_H__
#define __PROTOCOL_H__

enum protocol
{
    CAMERA,
    DC_MOTOR,
    BLDC_MOTOR,
    PMSM_MOTOR,
    ACIM_MOTOR,
    POWER_LED,
    I2C,
    SPI,
    LIDAR,
    CAN,
    ECAP,
    END
};

#define PROTOCOL_CALL_BUFFER_COUNT  (END)
#define PROTOCOL_CALL_BUFFER        ((END) - 1)

#endif
```

protocol.h File은

Project에서 사용 되는 통신 프로토콜의 리스트를

작성한 것으로 열거형 enum을 사용하여

각 프로토콜 종류마다 고유한 식별 번호를 할당한다.

맨 마지막의 END는 프로토콜 리스트의 끝을 알리며

프로젝트에 사용되는 프로토콜의 개수와 List Size를 계산하는데

이용할 수 있다.

이를 이용하여 프로토콜 리스트에 새로운 프로토콜이 추가되어도

별도로 코드를 수정하지 않고

변경된 프로토콜의 개수와 List Size 를 코드에 즉시 적용한다.

DDD Example Code (protocol/)

EDDI
Electronic Design Development Institute

2. protocol_call_table.h

```
#ifndef __PROTOCOL_CALL_TABLE_H__
#define __PROTOCOL_CALL_TABLE_H__

#include <stdio.h>
#include "protocol.h"

// void (*)(void) == protocol_call_ptr_type
// int (*)(int)

typedef void (*protocol_call_ptr_t) (void);

void protocol_not_implemented (void)
{
    printf("미구현 스펙 입니다.\n");
}

const protocol_call_ptr_t protocol_call_table[PROTOCOL_CALL_BUFFER_COUNT] = {
    [0 ... PROTOCOL_CALL_BUFFER] = &protocol_not_implemented,
    #include "protocol_call_table_mapper.h"
};

#endif
```

protocol_call_table.h File은

protocol.h File의 프로토콜 리스트를 바탕으로

함수 포인터 배열을 사용해 Protocol Handler Table을 작성한다.

이 테이블은 protocol_call_table로 명명하며

배열을 protocol_not_implemented 함수로 초기화 한다.

이후 protocol_call_table_mapper.h File을 이용하여

protocol_call_table에 프로토콜의 식별 번호에 맞춰

Handler를 할당하는데

이 때 할당되지 않은 Handler는

protocol_not_implemented 함수로 처리한다.

DDD Example Code (protocol/)

EDDI
Electronic Design
Development Institute

3. protocol_call_table_mapper.h

```
#ifndef __PROTOCOL_CALL_TABLE_MAPPER_H__
#define __PROTOCOL_CALL_TABLE_MAPPER_H__

#include "protocol_handler.h"

#define __PROTOCOL_CALL_TABLE(nr, sym) [nr] = sym,

__PROTOCOL_CALL_TABLE(CAMERA, proc_camera)
__PROTOCOL_CALL_TABLE(DC_MOTOR, proc_dc_motor)
__PROTOCOL_CALL_TABLE(BLDC_MOTOR, proc_bldc_motor)
__PROTOCOL_CALL_TABLE(PMSM_MOTOR, proc_pmsm_motor)
__PROTOCOL_CALL_TABLE(ACIM_MOTOR, proc_acim_motor)
__PROTOCOL_CALL_TABLE(POWER_LED, proc_power_led)
__PROTOCOL_CALL_TABLE(I2C, proc_i2c)
__PROTOCOL_CALL_TABLE(SPI, proc_spi)
__PROTOCOL_CALL_TABLE(LIDAR, proc_lidar)
__PROTOCOL_CALL_TABLE(CAN, proc_can)
__PROTOCOL_CALL_TABLE(ECAP, proc_ecap)

#endif
```

protocol_call_table_mapper.h File은

protocol_call_table.h File의

Protocol Handler Table(protocol_call_table)을 위해

프로토콜 리스트의 식별 번호에 맞게 처리하기 위한

protocol_handler.h File 에 정의되어 있는

Handler Method를 할당한다.

DDD Example Code (protocol/)

EDDI
Electronic Design Development Institute

4. protocol_handler.h

```h
#ifndef __PROTOCOL_HANDLER_H__
#define __PROTOCOL_HANDLER_H__

void proc_camera (void);
void proc_dc_motor (void);
void proc_bldc_motor (void);
void proc_pmsm_motor (void);
void proc_acim_motor (void);
void proc_power_led (void);
void proc_i2c (void);
void proc_spi (void);
void proc_lidar (void);
void proc_can (void);
void proc_ecap (void);

#endif
```

5. protocol_handler.c

```c
#include "protocol_handler.h"

#include <stdio.h>
void proc_camera (void)
{
    printf("Camera 프로토콜을 처리합니다.\n");
}

void proc_dc_motor (void)
{
    printf("DC Motor 프로토콜을 처리합니다.\n");
}

void proc_bldc_motor (void)
{
    printf("BLDC Motor 프로토콜을 처리합니다.\n");
}

void proc_pmsm_motor (void)
{
```

protocol_handler Library는 protocol_call_table.h File의 Protocol Handler Table(protocol_call_table)을 위해

프로토콜 리스트의 식별 번호에 맞게 이를 처리하는 Handler Method를 포함한다.

DDD Example Code (receiver/)

EDDI
Electronic Design
Development Institute

1. receiver.h

```h
#ifndef __RECEIVER_H__
#define __RECEIVER_H__

void recv_command_from_outbound (void);

#endif
```

2. receiver.c

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

#include "receiver.h"

#include "../protocol/protocol_handler.h"
#include "../protocol/protocol_call_table.h"

void recv_command_from_outbound (void)
{
    int command;

    printf("외부에서 명령이 들어온다 가정(랜덤 생성)\n");

    srand(time(NULL));

    command = rand() % 11;
    printf("수신된 명령 : %d\n", command);

    protocol_call_table[command]();
}
```

receiver Library는

Project의 외부 통신 프로토콜 수신을 모사하기 위해 작성한 것으로

srand(time(NULL)) 함수로 생성한 seed (rand) 를 바탕으로 생성한

난수 값 (command)을 printf 함수를 사용하여 출력하고,

함수 포인터 배열 protocol_call_table [command]는

Protocol.h File에 작성된 프로토콜 리스트를 바탕으로

Protocol_call_table_mapper.h 를 통해 Mapping 된

command 번째 요소의 Handler Method를 실행한다.