

이순신 운영체제

사용자 설명서

목차

1. 본 문서는.....	3
2. 이순신 운영체제는?.....	4
2.1. 운영체제의 시작.....	4
2.2. 개발 컨셉.....	4
2.3. 적용된 프로젝트.....	5
2.4. 향후 계획.....	5
3. 기본 구성.....	6
3.1. 지원 MCU.....	6
3.2. IDE.....	6
4. Crossworks for ARM.....	7
4.1. Crossworks for ARM을 사용하는 이유?.....	7
4.2. 다운로드.....	7
4.3. 설치.....	7
4.4. 라이선스.....	8
4.5. 평가판.....	8
4.6. 패키지 설치.....	9
5. 부팅 과정.....	10
5.1. reset_handler.....	10
5.2. Startup Code.....	11

1. 본 문서는

이순신 운영체제의 사용 설명서로 운영체제의 모든 정보를 담고 수시로 업데이트가 될 예정이다.

이 문서에서 다루는 MCU의 코어들은 모두 ARM Cortex-M 계열 제품으로 별도의 설명이 없는 경우 ARM Cortex-M3, M4F, M7을 기준으로 한다.

2. 이순신 운영체제는?

2.1. 운영체제의 시작

2007년 정부 지원으로 MDS 아카데미에서 6개월간 임베디드 교육을 받던 때이다. ARM 아키텍처와 네오스라는 MDS에서 직접 만든 운영체제 수업을 듣던 중에 Context Switching(문맥 전환)을 하는 방법에 대해 강한 호기심을 갖기 시작했다.

처음에는 ATmega128(AVR 8bit)을 시작으로 AT32UC3(AVR 32bit), ARM Cortex-M 계열에서 동작하는 선점형 라운드 로빈 스케줄링을 사용하는 문맥 전환 코드를 구현해 나갔다.

JAVA의 GUI 프로그래밍에 강한 매력을 느끼고 그와 유사한 방식으로 MCU에서 사용 가능한 라이브러리를 만들어보고 싶었고, 그 결과 C++을 사용하여 유사한 느낌의 그래픽 라이브러리 구현에 성공하게 되었다.

2007년부터 2014년까지 개인적으로 공부해보는 수준에서 다루어 보았지만, 2015년에 처음 코드를 오픈 하게 되었다.

2.2. 개발 컨셉

모든 ARM Cortex-M 계열의 MCU들을 제조사에 관계 없이 동일한 환경에서 개발이 가능하도록 하는 것이다. 기본 내부 장치 드라이버를 모두 통일하고 MCU의 내부 뿐만 아니라 외부에 장착되는 부품들에 대한 라이브러리까지도 완벽하게 공유할 수 있도록 하는 것이다.

최근 MCU 제조사들이 내부 주변 장치에 대한 코드 생성기를 제공해 줌으로써 내부 주변 장치를 이전보다 쉽게 사용하는 것을 고려했다면, 이에 더 나아가 주변 부품들에 대해 단순한 포팅과 초기화만 해주면 쉽게 사용할 수 있도록 하는 것이다.

지원하는 모든 MCU는 동일한 그래픽 라이브러리를 지원하기 때문에 MCU가 바뀌더라도 이전에 작성한 GUI 코드를 그대로 활용이 가능하도록 하고 있다.

2.3. 적용된 프로젝트

os가 적용된 대표적인 프로젝트는 아래와 같다.

> 카드 프린터

열 전사 방식 컬러 카드 프린터의 인쇄 엔진과 프린터 운용을 동시에 처리하는 프로젝트에서 적용되었다. 2019년부터 양산이 진행되었다.

> 선박 엔진 모니터

선박의 ECU 엔진으로부터 들어오는 정보를 디스플레이 하는 프로젝트에서 적용되었다. 해당 제품은 2020년부터 양산이 진행되었다.

> 가스 엔진 스로틀 밸브

스피드 컨트롤러로부터 밸브의 열림 양을 받아, 제어하여 엔진의 RPM을 컨트롤 하는 장치로 현재 테스트 까지 진행되어 있다.

> 가스 엔진 믹서

람다 컨트롤러에서 넘어오는 열림 양을 받아, 제어하여 가스와 공기의 혼합 비율을 제어하는 장치로 현재 테스트 까지 진행되어 있다.

> AVR(Auto Voltage Regulator) 컨트롤러

비상 발전기용 디젤 엔진의 동체의 필드 전압을 제어하여 요구되는 AC 전압으로 조정하는 제품 개발에서 적용되었다. 해당 제품은 2021년부터 양산이 진행되었다.

앞으로도 계속 새로운 프로젝트에 적용될 예정이다.

2.4. 향후 계획

지속적인 업그레이드를 해 나아가고 이미 작성된 코드에 대해서도 최적화를 해 나갈 계획이다.

3. 기본 구성

3.1. 지원 MCU

자세한 지원 목록은 아래 주소를 참고하라.

<https://cafe.naver.com/yssoperatingsystem/384>

3.2. IDE

현재 사용 중인 IDE는 **Crossworks for ARM**으로 ST를 포함한 많은 MCU 제조사의 ARM Cortex MCU들을 지원하는 개발 환경이다. 리눅스, 윈도우, 맥을 지원한다.

현재 **Crossworks for ARM** 외의 다른 툴은 지원하지 않고 있지 않다.

4. Crossworks for ARM

4.1. Crossworks for ARM을 사용하는 이유?

그래픽 라이브러리 개발에 C++을 사용하기 원했으나 IAR, Keil 등의 툴에서 전역으로 선언된 객체의 생성자를 호출 해주는 startup code 부분이 제공이 안되었다. Crossworks for ARM에서는 해당 부분을 잘 지원 해주고 부가적으로 리눅스와 맥에서도 사용 가능한 장점이 있다.

이순신 운영체제는 ST사의 MCU만 지원 할 것이 아니라 다른 MCU들도 지원할 계획이다. MCU 제조사 전용 IDE가 아닌 범용 IDE가 필요 했고, 가격도 비교적 저렴한 편이다.

4.2. 다운로드

아래 URL을 클릭하여 Rowley Associates 홈페이지에 방문한다.

<https://www.rowley.co.uk/arm/index.htm>

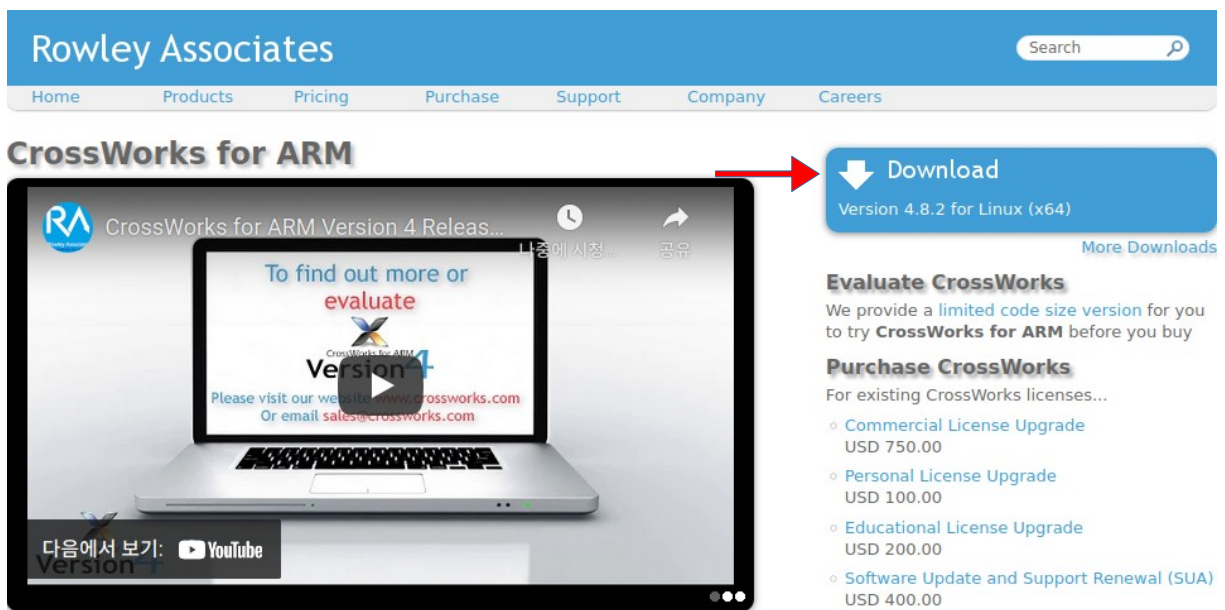


그림 1: 다운로드

위와 같이 Download 버튼을 클릭하여 설치 파일을 다운로드 받는다.

웹 페이지에 접속한 PC의 운영체제에 따라 **Download** 버튼의 다운로드 받을 프로그램의 OS 버전이 결정된다. 만약 다른 OS의 버전이나 이전 버전을 원할 경우, 버튼의 우측 하단에 나타난 **More Downloads** 글자를 클릭 한다.

4.3. 설치

Rowley Associates 홈페이지에 나타난 설치 방법을 참고 하라.

4.4. 라이선스

Crossworks for ARM의 라이선스는 네 가지가 있다.

- **개발자 공유 라이선스 (Shared Developer)**

모든 개발자가 USB 동글을 이용하여 라이선스를 공유하는 방식이다.

- **개발자 지정 라이선스 (Named Developer)**

지정된 사람에 대해 본인의 PC에 설치 갯 수에 관계 없이 사용하는 방식이다.

- **교육 기관용 라이선스 (Educational Workstation License)**

대학, 학교등의 교육 기관에서 라이선스 당 한 대의 PC에 설치 할 수 있는 방식이다.

- **개인용 라이선스 (Personal Non-Commercial License)**

개인이 비 상업적으로 사용할 목적으로 본인의 PC에 설치 갯 수에 관계 없이 사용하는 방식 이다.

비 상업적 사용이 불가능한 것 외에는 **개발자 지정 라이선스**와 동일하다.

라이선스는 웹에서 구입 가능하다. 개인용 라이선스의 경우 구입 절차 상에 비 상업적 사용에 관한 서약서를 작성해야 한다.

4.5. 평가판

라이선스 구입에 앞서 사용자는 1개월 평가판 사용이 가능하다. 경험에 의하면 약 2~3회 가량 한 메일 계정에 대하여 평가판 신청이 가능하다. 평가판은 유료 라이선스 버전과 동일하게 기능의 제약 없이 사용 가능하다.

기본적으로 라이선스가 설치되지 않으면, 16 kB까지 유료 라이선스와 동일하게 사용이 가능하다.

4.6. 패키지 설치

메뉴에서 Tools → Package Manager를 클릭 한다.

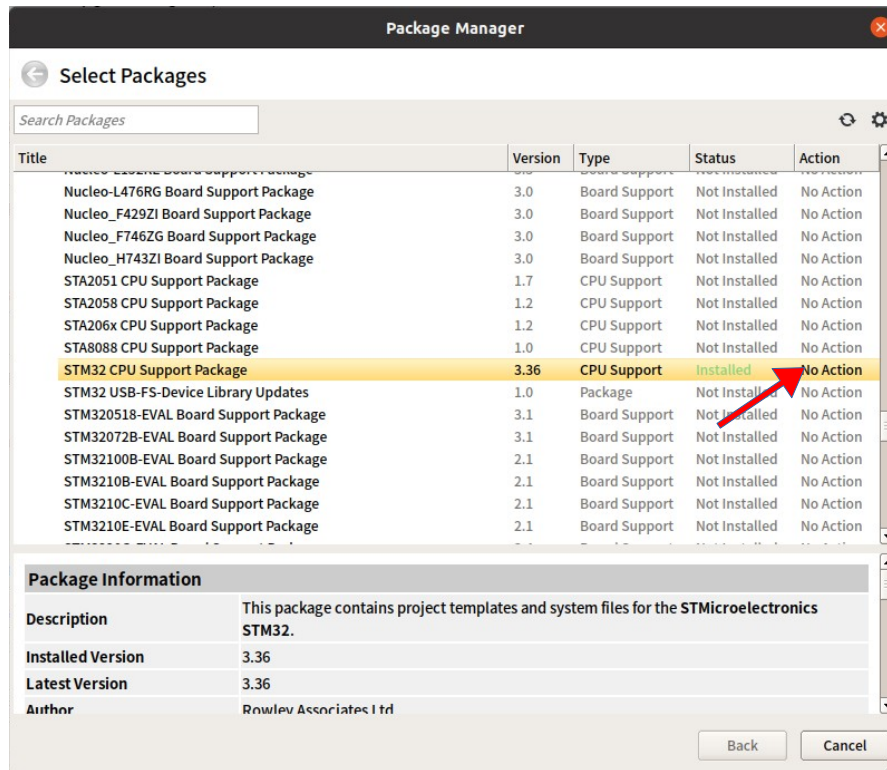


그림 2: 패키지 매니저

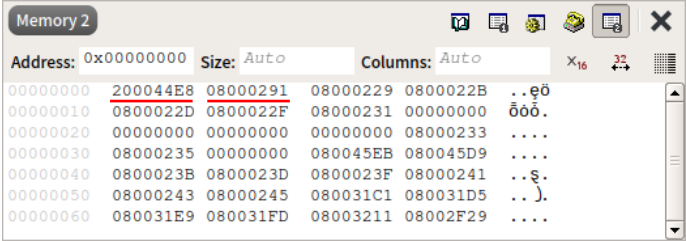
사용하고자 하는 CPU의 지원 패키지를 찾아 더블 클릭 하면 화살표가 가리키는 부분의 상태가 바뀐다. No Action은 아무것도 하지 않는 상태이고, 설치 되지 않은 패키지를 더블 클릭 할 경우 install로 상태가 바뀐다. 이때 하단에 나타나는 Next 버튼을 클릭하면 설치가 진행 된다.

CPU 지원 패키지가 설치되지 않은 경우 해당 MCU에 대해 사용이 불가능 하다.

5. 부팅 과정

5.1. reset_handler

ARM Cortex-M MCU는 리셋 신호가 발생되면 메모리의 0번지에 저장된 4바이트를 Stack Pointer(SP)로 가져오고, 4번지에 저장된 4바이트를 Program Counter(PC)로 가져온다.

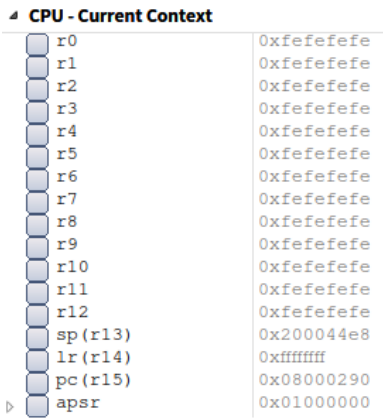


Address	Size	Columns
00000000	200044E8	08000291
00000010	0800022D	0800022F
00000020	00000000	00000000
00000030	08000235	00000000
00000040	0800023B	0800023D
00000050	08000243	08000245
00000060	080031E9	080031FD

그림 3: 메모리 저장 정보

위 그림 3에 나타난 0번지의 내용은 0x200044E8이고 4번지의 내용은 0x08000291이다. MCU에 리셋 신호가 발생하면 아래 그림 4과 같이 SP는 0x200044E8이 되고, PC는 0x08000290이 된다.

여기서 플래시 메모리에는 0x08000291 저장되어 있지만 PC에는 1이 빠진 0x08000290이 올라갔다. 그 이유는 호출되는 함수의 번지를 실제 메모리에 배치된 값으로 지정할 경우 MCU 코어는 ARM 명령어로 취급한다. 실제 메모리에 배치된 값에서 +1이 되어 있는 경우 Thumb 명령어로 취급한다. ARM Cortex-M은 Thumb 명령어만 지원하므로 모든 함수 포인터는 해당 번지의 +1 값을 갖고 있다.



Register	Value
r0	0xfefefefe
r1	0xfefefefe
r2	0xfefefefe
r3	0xfefefefe
r4	0xfefefefe
r5	0xfefefefe
r6	0xfefefefe
r7	0xfefefefe
r8	0xfefefefe
r9	0xfefefefe
r10	0xfefefefe
r11	0xfefefefe
r12	0xfefefefe
sp (r13)	0x200044e8
lr (r14)	0xffffffff
pc (r15)	0x08000290
apsr	0x01000000

그림 4: 리셋 후 레지스터

아래 그림 5와 같이 0x08000290 번지는 reset_handler() 함수가 배치되어 있다. reset_handler()가 본격적인 C/C++ 언어에서 main 함수가 시작되기 위한 메모리 설정 startup code의 시작이다.

```

113 reset_handler:|
    #ifndef __NO_SYSTEM_INIT
    ..ldr r0, __RAM_segment_end__
    ..mov sp, r0
    ..bl SystemInit
    #endif
120
    #ifdef VECTORS_IN_RAM
    ..ldr r0, __vectors_load_start__
    ..ldr r1, __vectors_load_end__
    ..ldr r2, __vectors_ram
    l0:

```

그림 5: reset_handler

5.2. Startup Code

Project Items	Code	Data+RO
gui 23 files		
inc 329 files		
instance 21 files		
mod 28 files		
mcp 11 files		
protocol 1 file		
sac 12 files		
scheduler 3 files		
stdlib 4 files		
system 8 files		
util 13 files		
System Files 3 files	[880]	[4]
STM32F103xG.vec		
STM32_Startup.s	476	
thumb_crt0.s	404	4
Output Files		

그림 6: startup code가 배치된 파일

위 그림 6과 같이 Project Explorer 창의 “System Files” 폴더의 STM32_Startup.s 파일에 reset_handler가 들어있다. 다른 MCU의 경우 파일 명에 차이가 있으나 같은 경로에 들어 있다.

```

reset_handler:
    #ifndef __NO_SYSTEM_INIT
    ldr r0, __RAM_segment_end__ // ①
    mov sp, r0                // ②
    bl SystemInit              // ③
    #endif

```

텍스트 1: SystemInit 호출

위 텍스트 1에 나타난 코드에서 ①은 Link Script 파일에 정의된 __RAM_segment_end__의 값을 r0에 읽어 들이는 코드이다. ②는 SP에 R0값을 이동 시키는 명령이다. 코드 ①, ②는 결과적으로 5.1.단원에 나타낸 0번지의 4바이트에 SP로 불러 들이는 내용과 동일한 역할을 하므로 무시해도 좋다. ③에 나타난 bl

명령은 함수 호출 명령이다. SystemInit의 이름을 갖는 함수를 호출한다. 링킹 과정에서 어셈블리의 레이블과 함수의 이름과 전역 변수명은 유사하게 취급이 된다. 현재 SystemInit이 어셈블리어로 작성되어 있든, C언어로 작성되어 있든 관계없이 해당 함수가 존재한다면 호출한다. 일반적으로는 SystemInit 함수는 C언어로 작성되어 있다.

이순신 OS에서는 아래 그림 7과 같이 “yss/msp” 경로에 SystemInit 함수가 배치된 파일들이 있다. 각 MCU 제조사 별로 폴더가 구성되고 제조사에 대해 패키지 별로 파일이 구성되어 있다.

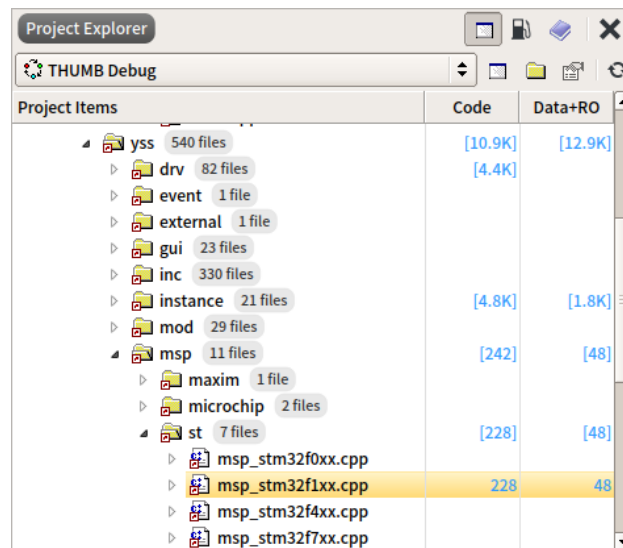


그림 7: msp 파일 경로

SystemInit 함수의 주 목적은 Startup Code에 진입하기 전에 클럭 설정과 메모리 설정을 먼저 하기 위해서이다. 이 과정이 끝나면 본격적인 Startup Code가 시작된다.

아래 텍스트 2는 SystemInit 함수 호출 후, FPU를 활성화 시키는 코드이다. MCU 타입(FPU가 없는 MCU)이나 설정에 의해 이 코드 블록은 무효화 될 수 있다.

MCU에 FPU가 내장되어 있어도 프로젝트 기본 설정은 비활성으로 되어 있다. IDE의 **Project Properties**에서 **Code Generation** 옵션에 가면 **ARM FP ABI Type** 항목에서 바꿀 수 있다.

```

#if !defined(__NO_FPU) && !defined(__SOFTFP__)
// Enable CP11 and CP10 with CPACR |= (0xf<<20)
movw r0, 0xED88 // ①
movt r0, 0xE000 // ②
ldr r1, [r0] // ③
orrs r1, r1, #(0xf << 20) // ④
str r1, [r0] // ⑤
#endif
#if !defined(__NO_RUNFAST_MODE)
lsb // ⑥
dsb // ⑦
vmrs r0, fpscr // ⑧
orrs r0, r0, #(0x3 << 24) // FZ and DN
vmsr fpscr, r0 // ⑨
// clear the CONTROL.FPCA bit
mov r0, #0 // ⑩
msr control, r0 // ⑪
// FPDSCR similarly
movw r1, 0xEF3C // ⑫
movt r1, 0xE000 // ⑬
ldr r0, [r1] // ⑭
orrs r0, r0, #(0x3 << 24) // FZ and DN
str r0, [r1] // ⑮
#endif
#endif

```

텍스트 2: FPU 설정

- ① 라인은 r0의 하위 16비트에 0xED88 값을 넣는 코드이다.

레지스터	값
r0	0x 0000 ED88

표 1: 레지스터

- ② 라인은 r0의 상위 16비트에 0xE000 값을 넣는 코드이다.

레지스터	값
r0	0x E000 ED88

표 2: 레지스터

- ③ 라인은 0xE000ED88 번지의 데이터를 r1 레지스터