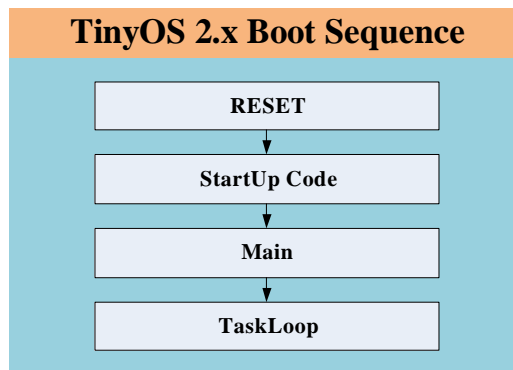


# TinyOS 2.x Boot Sequence

본 기술문서에서는 TinyOS 2.x의 부트 과정에 대하여 알아본다. Atmega128 프로그램 메모리는 크게 어플리케이션 영역과 Boot Loader 영역으로 나누어지며, 전원의 공급이 이루어지면 MCU는 프로그램 메모리의 0번지에 있는 코드부터 수행한다. Atmega128에서는 인터럽트 벡터 테이블이 프로그램 메모리의 0번지부터 위치하게 되며 어플리케이션 영역이다. 참고로, Atmega128의 프로그램 메모리는 Boot Loader 영역과 어플리케이션 영역으로 나누어지지만 (네트워크 Re-programming 문서 참조) Atmega128의 Boot Loader 영역은 TinyOS 2.x의 부팅 과정과는 전혀 상관 없으며, 단지 In System Flash 메모리의 Re-programming에만 사용된다. 프로그램 메모리의 0번지부터 main() 함수로 점프하기 전까지의 코드는 프로그래머가 작성한 코드가 아니라 AVR 컴파일러가 Atmega128을 위해 작성한 StartUp 코드이다. 이 StartUp 파일은 컴파일러에 의해 자동으로 링크 된다.



TinyOS 2.x의 부트 과정은 위의 그림처럼 4단계로 이루어진다.

## ■ RESET

인터럽트 벡터 테이블의 주소로 점프하여 StartUp Code가 실행되도록 한다.

## ■ StartUp Code

상태 레지스터를 초기화하고 스택포인터를 RAMEND (데이터 메모리의 마지막 주소)로 설정한다. 그리고 .data 세그먼트와 .bss 세그먼트를 초기화한 후 main() 함수로 점프한다.

## ■ Main

스케줄러, 플랫폼, 소프트웨어 초기화를 수행하고, booted 이벤트를 발생시킨 후 태스크 루프로 분기한다.

- TaskLoop

태스크가 존재하면 수행하고, 그렇지 않으면 MCU를 sleep 상태로 전환하는 무한 태스크 루프이다.

## 1. StartUp Code

StartUp 파일을 이용하여 0번지부터 작성된 코드는 아래와 같은 작업을 수행한다.

- 상태 레지스터 초기화
- 스택포인터를 RAMEND로 설정(Atmega128의 경우 SRAM의 끝번지인 10FF로 설정)
- 전역변수를 위한 메모리 공간 할당 (0x0100부터 시작) 및 .data 세그먼트(초기 값을 가지는 전역 변수)와 .bss 세그먼트(초기 값을 가지지 않는 전역 변수) 초기화. 여기서 .data 세그먼트 영역의 전역변수 초기값은 프로그램 Flash 메모리의 마지막 부분에 저장되어있으며 .bss 세그먼트 영역의 전역변수는 0으로 초기화된다.
- main() 함수로 점프

원래 AVR 컴파일러는 StartUp Code에 대해서 바이너리 실행 이미지로 제공된다. 본 기술문서에서는 AVR Studio를 이용하여 StartUp Code 코드를 disassemble하여 분석한다.

```

+00000000: 940C0046 JMP 0x00000046 Jump
+00000002: 940C0063 JMP 0x00000063 Jump
+00000004: 940C0063 JMP 0x00000063 Jump
+00000006: 940C0063 JMP 0x00000063 Jump
+00000008: 940C0063 JMP 0x00000063 Jump
+0000000A: 940C0063 JMP 0x00000063 Jump
+0000000C: 940C0063 JMP 0x00000063 Jump
+0000000E: 940C0063 JMP 0x00000063 Jump
+00000010: 940C0063 JMP 0x00000063 Jump
+00000012: 940C0063 JMP 0x00000063 Jump
+00000014: 940C0063 JMP 0x00000063 Jump
+00000016: 940C0063 JMP 0x00000063 Jump
+00000018: 940C0063 JMP 0x00000063 Jump
+0000001A: 940C0063 JMP 0x00000063 Jump
+0000001C: 940C0063 JMP 0x00000063 Jump
+0000001E: 940C0063 JMP 0x00000063 Jump
+00000020: 940C0063 JMP 0x00000063 Jump
+00000022: 940C0063 JMP 0x00000063 Jump
+00000024: 940C0063 JMP 0x00000063 Jump
+00000026: 940C0063 JMP 0x00000063 Jump
+00000028: 940C0063 JMP 0x00000063 Jump
+0000002A: 940C0063 JMP 0x00000063 Jump
+0000002C: 940C0063 JMP 0x00000063 Jump
+0000002E: 940C0063 JMP 0x00000063 Jump
+00000030: 940C0063 JMP 0x00000063 Jump
+00000032: 940C0063 JMP 0x00000063 Jump
+00000034: 940C0063 JMP 0x00000063 Jump
+00000036: 940C0063 JMP 0x00000063 Jump
+00000038: 940C0063 JMP 0x00000063 Jump
+0000003A: 940C0063 JMP 0x00000063 Jump
+0000003C: 940C0063 JMP 0x00000063 Jump
+0000003E: 940C0063 JMP 0x00000063 Jump
+00000040: 940C0063 JMP 0x00000063 Jump
+00000042: 940C0063 JMP 0x00000063 Jump
+00000044: 940C0063 JMP 0x00000063 Jump
+00000046: 2411 CLR R1 Clear Register

```

위의 그림을 참고로 우리는 다음과 같은 사실을 확인 할 수 있다.

- 가장 먼저 수행되는 0번지는 RESET 벡터이며 RESET 인터럽트 벡터 테이블은 Startup Code의 첫 번째 주소 (0x00000046)를 가지고 있다. 참고로 Atmega128의 인터럽트 벡터 테이블은 총 36개의 4바이트 엔터티를 가진다.
- 각 인터럽트에 대한 ISR(Interrupt Service Routine)이 정의되면 위 그림에서 사각형으로 표시된 것처럼 ISR의 주소로 설정됨

```

00000046: 2411 CLR R1 Clear Register
00000047: BE1F OUT 0x3F,R1 Out to I/O location
00000048: EFCF SER R28 Set Register
00000049: E1D0 LD1 R29,0x10 Load immediate
0000004A: BFDE OUT 0x3E,R29 Out to I/O location
0000004B: BFCD OUT 0x3D,R28 Out to I/O location

```

[코드 1]

- [코드 1]
  - 상태 레지스터(SREG:0x3F)의 값을 0으로 초기화
  - R28(L), R29(H) 레지스터(Y 레지스터)의 값을 설정하여 스택 포인터를 RAMEND(0x10FF)로 설정

0000004C:	E011	LDI	R17,0x01	Load immediate
0000004D:	E0A0	LDI	R26,0x00	Load immediate
0000004E:	E0B1	LDI	R27,0x01	Load immediate
0000004F:	EDEC	LDI	R30,0xDC	Load immediate
00000050:	E0F0	LDI	R31,0x00	Load immediate

[코드 2]

■ [코드 2]

- R26(L), R27(H) 레지스터(X 레지스터)의 값을 0x0100(data 메모리의 시작주소)로 설정
- R30(L), R31(H) 레지스터(Z 레지스터)의 값을 0x00DC로 설정
- Z 레지스터는 .data 세그먼트 초기화를 위해 프로그램 메모리에 저장된 초기값의 시작 주소를 가짐

00000051:	E000	LDI	R16,0x00	Load immediate
00000052:	BF0B	OUT	0x3B,R16	Out to I/O location

[코드 3]

■ [코드 3]

- 램 페이지 결정 레지스터(RAMPZ:0x3B)의 값을 0으로 초기화

00000053:	C002	RJMP	PC+0x0003	Relative jump
00000054:	9007	ELPM	R0,Z+	Extended load program memory and postincrement
00000055:	920D	ST	X+,R0	Store indirect and postincrement
00000056:	30A8	CPI	R26,0x08	Compare with immediate
00000057:	07B1	CPC	R27,R17	Compare with carry
00000058:	F7D9	BRNE	PC-0x04	Branch if not equal

[코드 4]

■ [코드 4]

- 프로그램 메모리에서 초기 값을 로드하여 .data 세그먼트를 초기화
- ELPM 명령으로 Z 레지스터가 가리키는 프로그램 메모리 주소에 저장된 값을 R0 레지스터에 저장하고 다음 초기 값을 가리키도록 Z 레지스터의 값을 증가
- R0의 값을 .data 세그먼트에 저장하고 X 레지스터의 값을 증가
- .data 세그먼트의 초기화가 완료될 때까지 반복

00000059:	E011	LDI	R17,0x01	Load immediate
0000005A:	E0A8	LDI	R26,0x08	Load immediate
0000005B:	E0B1	LDI	R27,0x01	Load immediate

[코드 5]

■ [코드 5]

- X 레지스터의 값을 .data 세그먼트 다음 주소로 설정

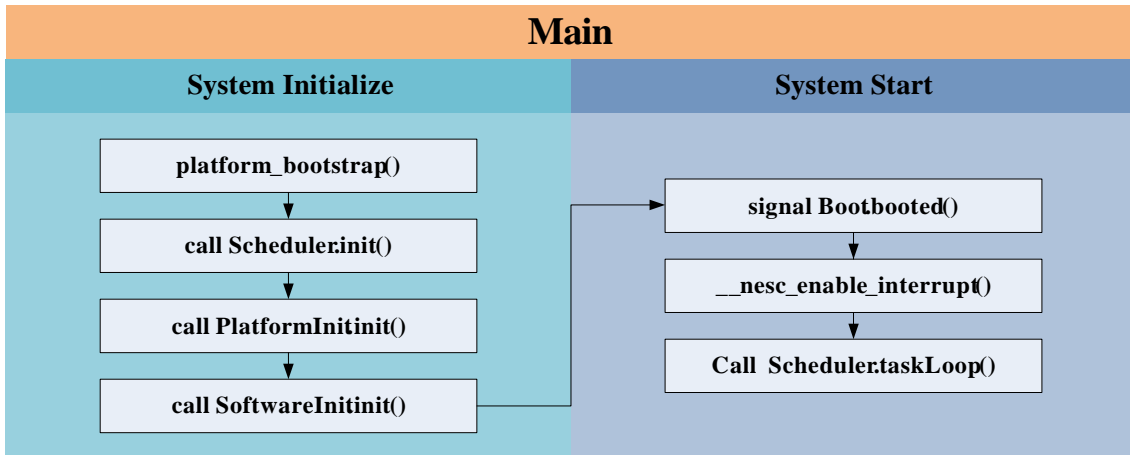
0000005C:	C001	RJMP	PC+0x0002	Relative jump
0000005D:	921D	ST	X+,R1	Store indirect and postincrement
0000005E:	31A0	CPI	R26,0x10	Compare with immediate
0000005F:	07B1	CPC	R27,R17	Compare with carry
00000060:	F7E1	BRNE	PC-0x03	Branch if not equal
00000061:	940C0065	JMP	0x00000065	Jump
00000063:	940C0000	JMP	0x00000000	Jump
00000065:	main			

[코드 6]

■ [코드 6]

- .bbs 세그먼트 초기화
- .bbs 세그먼트의 크기만큼 0으로 초기화
- .data 세그먼트와 .bbs 세그먼트 초기화 완료 후 main() 함수로 점프

## 2. Main



위의 그림에서 기술된 바와 같이, TinyOS 2.x에서는 RealMainP 모듈의 main() 함수가 표준 부트 시퀀스를 수행하며 시스템을 초기화하는 부분과 시작하는 부분으로 구분된다.

다음은 RealMainP 모듈의 코드를 나타내며 시스템 초기화 및 시작을 수행한다.

```
atomic {
platform_bootstrap();
call Scheduler.init();
call PlatformInit.init();
while (call Scheduler.runNextTask());
call SoftwareInit.init();
while (call Scheduler.runNextTask());
}
```

```
__nesc_enable_interrupt();
signal Boot.booted();
call Scheduler.taskLoop();
return -1;
```

### ■ 시스템 초기화

#### • platform\_bootstrap()

가장 처음 호출되는 함수로 시스템이 실행상태로 들어가는 최소한의 기능이 정의 (메모리 시스템 구성 또는 프로세서 모드 설정)된 함수이다.

TinyOS 2.x의 최상위 include 파일인 tos.h에서 함수원형만을 정의하고 있으며 필요에 따라 platform.h에서 함수를 override하여 사용할 수 있다.

- Scheduler.init()

초기화를 수행하면서 태스크를 post하는 작업이 필요할 경우를 대비하여 스케줄러를 먼저 초기화 한다.

- PlatformInit.init()

사용된 플랫폼의 I/O 포트 및 각종 hardware 초기화를 수행한다(I/O pin 구성, Clock calibration, LED 구성 등).

- SoftWareInit.init()

이 함수는 여러 컴포넌트에 걸쳐서 광범위하게 수행되며 시스템이 시작할 때 각 모듈에서 사용하는 변수들을 초기화한다.

#### ■ 시스템 시작

- \_\_nesc\_enable\_interrupt()

시스템 초기화가 완료되면 글로벌 인터럽트를 가능하도록 설정한다.

- booted()

Boot.booted()를 signal하여 Boot 인터페이스를 사용하는 최상위 모듈에서 booted() 이벤트가 발생하도록 한다. 최상위 모듈의 booted() 이벤트에는 하위 컴포넌트의 시작을 정의한다. 예를 들어, 타이머를 1초마다 fire 되도록 startPeriodic(1024) 커맨드를 호출한다.

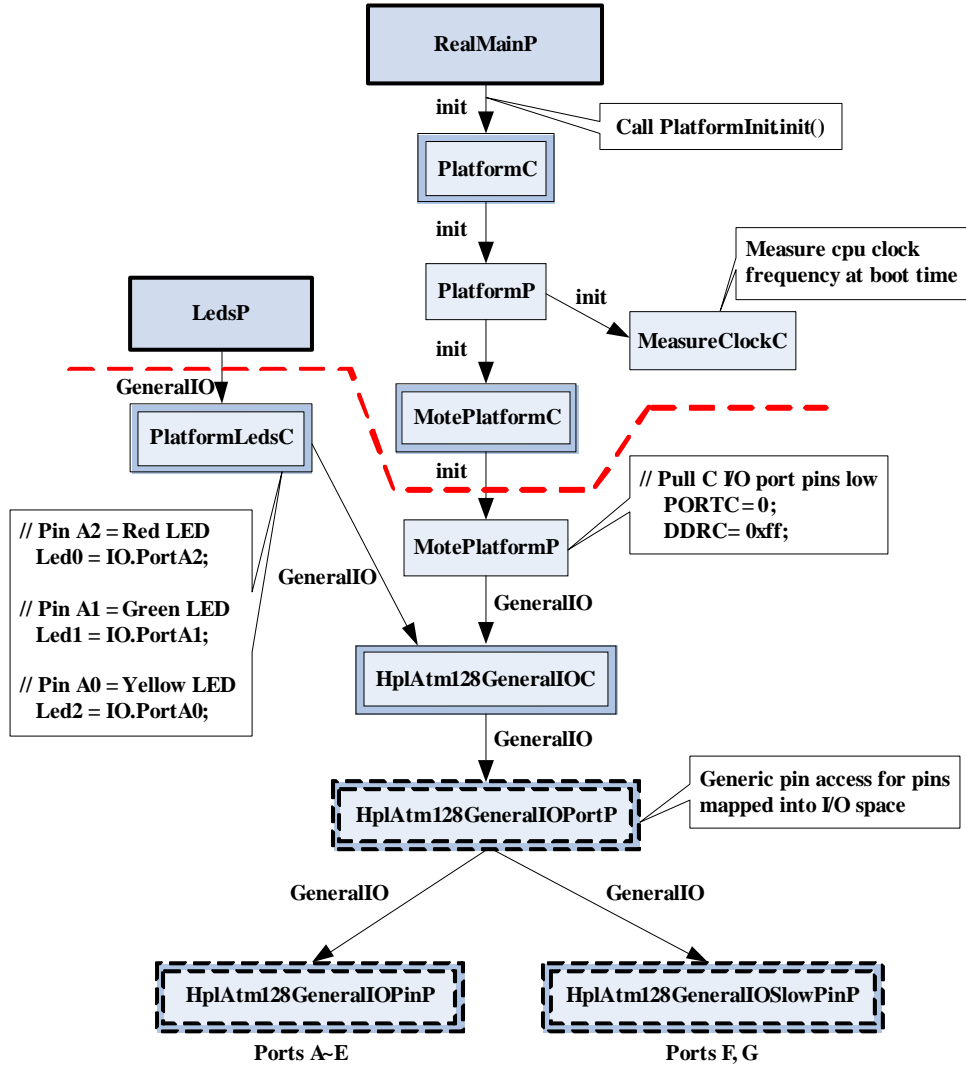
- taskLoop()

booted() 이벤트를 통해 시스템 시작이 완료되면 스케줄러는 태스크 루프 상태로 들어간다. 태스크 루프는 태스크 큐에 작업이 존재하면 해당 작업을 수행하고 수행할 태스크가 존재하지 않으면 MCU를 sleep 상태로 전환한다.

### 2.1 PlatformInit.init()

Blink 어플리케이션의 초기화 단계에서 PlatformC 컴포넌트의 init() 커맨드가 호출되면 LED 장치를 위한 I/O핀 매핑이 수행된다. 아래 그림은 RealMainP 컴포넌트에서 I/O핀 매핑을 수행하는 컴포넌트까지의 연결 상태를 나타낸다. 실제로 I/O핀 매핑에 관련된 부분은 점선 아래 컴포넌트들에 의해 수행된다. 각 컴포넌트는 I/O핀 매핑을 위해 GeneralIO 인터페이스를 사용한다. hplAtm128GeneralIOC 컴포넌트의 하위 컴포넌트들은 generic 컴포넌트로 각 장치에 대한 I/O핀 매핑 코드를 각각 생성한다. A~E 포트와 F, G포트는 서로 다른 컴포넌트로 연결되며 전자의 경우는 generic 컴포넌트가 GPIO핀의 8비트 포트로 연결되며 후자의 경우 ADC, WE, RD 등의 신호와 연결된다. F, G포트의 경우 extended I/O space에 매핑되며 sbi(),

cbi()를 사용할 수 없다. 실제로 Blink 어플리케이션에서는 포트 A와 C 레지스터만 값을 설정한다.

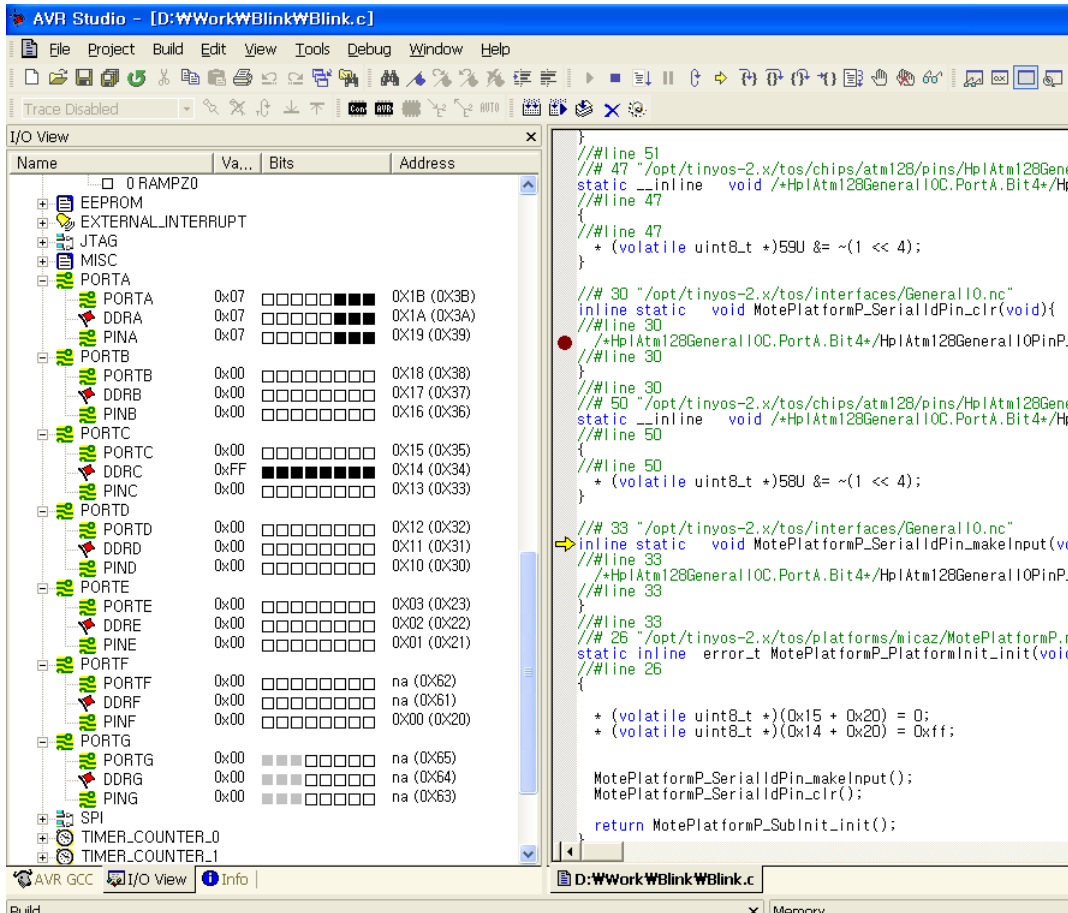


아래의 그림은 포트 A~G까지의 레지스터 상태를 나타내며 Blink 어플리케이션에서 요구하는 3개의 LED를 위해 포트 A를 사용한다. 3개의 LED는 메모리 0x3B에 다음과 같이 연결된다.

- A2 = Red LED
- A1 = Green LED
- A0 = Yellow LED

초기 상태에는 레지스터(PORTA 0x3B)의 하위 3비트를 1로 초기화하여 LED를 끈 상태를 유지한다. 그리고 외부 장치의 power 컨트롤을 위해 포트 C DDR(Data

Direction Register)를 0xff로 초기화하여 포트 C를 출력 전용으로 동작하도록 설정한다.



아래의 코드는 LED 장치에 대한 I/O 포트 레지스터를 설정하기 위한 함수이다.

```
static inline error_t LedsP_Init_init(void)
{
    LedsP_Led0_makeOutput(); // A2 DDRA 설정
    LedsP_Led1_makeOutput(); // A1 DDRA 설정
    LedsP_Led2_makeOutput(); // A0 DDRA 설정
    LedsP_Led0_set(); // A2 PORTA를 설정
    LedsP_Led1_set(); // A1 PORTA를 설정
    LedsP_Led2_set(); // A0 PORTA를 설정
    return SUCCESS;
}
```

여기서 LED0에 대해 I/O 레지스터 값을 쓰는 코드를 살펴본다. makeOutput() 함수가 호출되면 다시 HoIAtm128GeneralIOPinP 컴포넌트의 makeOutput() 함수가 호

출된다.

```
inline static void LedsP_Led0_makeOutput(void){
    HplAtm128GeneralIOPinP_2_IO_makeOutput();
}
```

메모리 58(=0x3A)의 값과 4를 OR연산한다. 실제로 레지스터에 값이 쓰여진다.

```
static __inline void HplAtm128GeneralIOPinP_2_IO_makeOutput(void)
{
    * (volatile uint8_t *)58U |= 1 << 2;
}
```

set() 함수가 호출되면 다시 HplAtm128GeneralIOPinP 컴포넌트의 set() 함수가 호출된다.

```
inline static void LedsP_Led0_set(void){
    HplAtm128GeneralIOPinP_2_IO_set();
}
```

메모리 59(=0x3B)의 값과 4를 OR연산한다. 실제로 레지스터에 값이 써진다.

```
static __inline void HplAtm128GeneralIOPinP_2_IO_set(void)
{
    * (volatile uint8_t *)59U |= 1 << 2;
}
```

## 2.2 SoftwareInit.init()

하드웨어 자원에 직접적으로 의존하지 않는 컴포넌트는 SoftwareInit() 함수와 연결되어 초기화를 수행한다. 예를 들어, TimerMilliP 같은 configuration은 Init 인터페이스를 제공하지 않고 MainC 컴포넌트를 HIL 컴포넌트와 이 함수로 직접 연결하여 시스템 초기화 루틴에서 필요한 초기화 작업이 수행 될 수 있도록 한다. 이러한 configuration을 "auto-wire" 되었다고 한다. (사용자에게 transparency를 제공) PlatformInit와 SoftwareInit를 수행하는 과정에서 태스크를 post하는 작업이 수행되었을 수도 있기 때문에 Scheduler.runNextTask()를 호출하는 루프를 이용하여 post된 태스크를 수행하고, 수행할 태스크가 없으면 루프를 빠져나와 다음 작업을 수행하도록 한다.