KIPS Trans. Comp. and Comm. Sys.
Vol.12, No.4 pp.127~134
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2023.12.4.127

Porting gcc Based eCos OS and PROFINET Communication Stack to IAR

Jin Ho Kim†

ABSTRACT

This paper describes how to port the eCos operating system and PROFINET communication stack developed based on gcc to the IAR compiler. The eCos operating system provides basic functions such as multi-thread, TCP/IP, and device driver for PROFINET operation, so there is no need to change it when developing PROFINET applications. Therefore, in this study, we reuse an eCos library built with gcc and it link with PROFINET communication stack that are ported to IAR complier. Due to the different of the gcc and IAR linker, symbol definitions and address of the constructors should be changed using the external tool that generates symbol definitions and address of the constructors from MAP file. In order to verify the proposed method, it was confirmed that the actual I/O was operating normally through PROFINET IRT communication by connecting to the Siemens PLC. IAR compiler has better performance in both the compile time and the size of the generated binary. The proposed method in this study is expected to help port various open sources as well as eCos and PROFINET communication stacks to other compilers.

Keywords : PROFINET, eCos, IAR, gcc, Industrial Ethernet

gcc 기반 eCos 운영체제 및 PROFINET 통신 스택의 IAR 포팅 방법

김 진 호†

요 약

본 논문에서는 gcc 기반으로 개발된 eCos 운영체제 및 PROFINET 통신 스택을 IAR 컴파일러로 포팅하는 방법에 대해 설명한다. eCos 운영체제의 경우 PROFINET 구동을 위한 멀티 스레드, TCP/IP, 디바이스 드라이버 등의 기반 기능을 제공하고 있어, PROFINET 어플리케이션 개발 시 변경할 필요가 없다. 따라서, 본 연구에서는 eCos는 gcc로 빌드된 라이브러리를 활용하고, 개발시 변경이 필요한 PROFINET 통신 스택은 IAR 로 포팅하여 함께 링킹하는 방안을 제안한다. IAR 링커와 gcc 링커의 차이로 인해 일부 섹션의 주소를 정의하는 심볼과 생성자의 주소가 정상적으로 생성되지 못하는 문제가 있어, MAP 파일을 읽어 해당 심볼 및 주소를 저장하는 외부 툴을 개발하였으며, 이 툴과 연동하여 동작할 수 있도록 부트로더의 소스 코드를 수정하였다. 제안하는 방법을 검증하기 위해 실제 지멘스 사의 PLC와 연결하여 PROFINET IRT 통신으로 실제 I/O 가 정상 동작하는지 검증하였으며, IAR 컴파일러가 컴파일 시간 및 생성된 바이너리 크기 모두 더 좋은 성능을 가지고 있음을 확인하였다. 본 연구에서 제안하는 방법은 eCos 및 PROFINET 통신 스택뿐 아니라 다양한 오픈 소스를 상용 컴파일러로 포팅하는데 도움을 줄 것으로 기대한다.

키워드 : 프로피넷, 이코스, IAR, gcc, 산업용 이더넷

1. 서 론

스마트팩토리 기술의 발전으로 기존 CAN 또는 RS-485 통신 기반의 저속 필드버스에서 고속인 실시간 Ethernet 프로토콜로 변경되고 있다[1-2]. 산업용 실시간 Ethernet은 주로 PLC (Programmable Logic Controll) 등의 마스터와 모터 드라이브 또는 I/O (Input/Output)와 같은 슬레이브 장치를 실시간으로 연결하는 역할을 수행한다. 이와 같은 실시간 Ethernet은 Siemens, Bechhoff, 미쯔비시, Rockwell 등 주요 제조사 별로 PROFINET, EtherCAT, CC-LINK IE, Ethernet/IP 등 전용의 통신 프로토콜을 개발하였으며[3], 상기 제조사들은 자사의 통신 프로토콜의 시장 점유율을 높이기 위해 노력하고 있다.

이와 같은 필드 버스에서 실시간 Ethernet 프로토콜로의 전환에 따라 기존 필드 버스 기반의 모터 드라이브 및 I/O 등의 슬레이브 장치는 고속 실시간 Ethernet을 지원할 수

※ 이 연구결과물은 2019년도 경남대학교 신진교수연구비 지원에 의한 것임.
† 종신회원 : 경남대학교 컴퓨터공학부 조교수
Manuscript Received : October 7, 2022
Accepted : December 28, 2022
* Corresponding Author : Jin Ho Kim(kimjh@kyungnam.ac.kr)

Table 1. Comparison of the Industrial Ethernet Protocols

	Manufacturer	Association	Support Com. stack
PROFINET	Siemens	PI	O
EtherCAT	Bechhoff	ETG	O
CC-LINK IE	Mitsubishi	CLPA	O
Ethernet/IP	Rockwell	ODVA	O

있도록 개발되어야 하며[4-9], 이를 위해 통신 프로토콜을 개발한 주요 제조사들은 자사의 통신 프로토콜을 쉽게 개발할 수 있도록 협회를 만들어 지원하고 있다. Table 1에서와 같이 대다수의 통신 프로토콜들은 협회를 통해 다양한 홍보 및 지원을 진행하고 있으며, 자사의 제품에 쉽게 적용할 수 있도록 통신 스택을 지원하고 있다. 본 논문에서 설명하는 PROFINET의 경우 제조사인 지멘스에서 평가 보드 기반으로 동작하는 PROFINET 통신 스택[10]을 제공하고 있다. 제공하는 PROFINET 통신 스택은 다양한 고객이 쉽게 사용할 수 있도록 무상인 gcc를 기반으로 개발되어 제공된다.

대표적인 오픈소스인 gcc는 다양한 플랫폼을 지원하는 컴파일러로 무료로 사용할 수 있는 장점에도 불구하고 실시간 및 신뢰성이 중요한 모터 드라이브 등의 자동화 장치 개발에는 널리 사용되고 있지 않다. 기존의 필드 버스 기반의 모터 드라이브 및 I/O 장치는 특정 MCU (Micro Controller Unit) 또는 DSP (Digital Signal Processor) 전용으로 개발된 상용 컴파일러를 사용하여 개발하는 경우가 많다. 컴파일러는 SW 전체의 성능과 신뢰성에 영향을 미치기 때문에 실시간 및 신뢰성이 중요한 자동화 장치의 개발 과정에서 컴파일러를 변경하는 경우는 매우 드물다. 따라서, 기존의 필드 버스를 새로운 실시간 이더넷으로 개발하기 위해서는 새로운 기능인 실시간 이더넷을 기존에 사용하고 있는 컴파일러로 개발하여 통신 기능을 검증하고, 이후에 검증된 기존의 SW를 재사용하여 개발하는 방향이 바람직하다. 따라서, 본 연구에서는 gcc 기반의 PROFINET 통신 스택을 상용 컴파일러인 IAR Studio 로 포팅하는 방법을 설명한다. 본 연구에서는 IAR 이라는 특정 컴파일러로의 포팅을 설명하고 있으나, 이를 통해 gcc 기반의 다양한 오픈소스 프로그램을 다양한 상용 컴파일러로 포팅하는 방법을 알 수 있을 것으로 예상한다.

Siemens 사에서 제공하는 PROFINET 통신 스택은 eCos [11]라는 오픈소스 기반의 실시간 운영체제 기반으로 동작하며, eCos는 PROFINET 통신 동작을 위해 필요한 멀티 스레드 환경, TCP/IP 통신 스택, printf 등의 표준 C 라이브러리, 디바이스 드라이버 등의 기반 환경 제공한다. 일반적으로 PROFINET 기반의 장치를 개발하는 과정에서 eCos는 초기 설정을 변경할 필요가 없고, eCos가 gcc에 맞게 개발되고 검증된 운영체제임을 감안하여 본 연구에서 eCos는 gcc로 컴파일된 오브젝트 파일을 그대로 사용하고, 변경이 필요한 PROFINET 통신 스택만 IAR로 포팅하는 방법을 제안한다. 제안하는 방법은 gcc 로 컴파일된 eCos 오브젝트 파일이 elf 표준[12]에 따라 생성되어 IAR 링커와 같이 elf 표준을 따르는 다른 링커에서 별도의 수정 없이 링킹이 가능한 특성을 이용한다. 다만, 이 과정에서 IAR 링커와 gcc 링커의 차이로 인해 일부 소스 코드 및 링커 스트립트의 수정이 필요하다. 특히, C++ 생성자 호출과 관련하여 IAR 링커가 생성자 함수의 주소를 정상적으로 생성하지 못해, 별도의 외부 툴을 이용하여 MAP 파일에서 생성자의 주소를 찾아 처리해주는 방법을 사용하였다. 세부 포팅 방법에 대한 내용은 본문에서 설명하고자 한다.

본 논문은 2장에서 gcc 기반의 PROFINET 개발 환경에 대해 설명하고, 3장에서 gcc 기반의 PROFINET 통신 스택을 IAR 컴파일러로 포팅하는 방법을 설명한다. 4장에서는 IAR로 포팅된 PROFINET 통신 스택을 검증하기 위한 검증 환경 및 검증 결과를 설명하고, 5장에서는 결론과 함께 논문을 마무리 한다.

2. PROFINET 개발 환경

본 장에서는 지멘스가 제공하는 gcc 기반의 PROFINET의 개발 환경을 이해하기 위한 평가 보드, PROFINET 통신 스택 및 개발 환경에 대해 설명한다.

2.1 PROFINET 평가 보드

PROFINET 평가 보드는 ERTEC 200P-2 라는 Siemens 사에서 개발한 ARM926EJ-S 기반의 MPU (Micro Processor Unit)를 사용한다. ERTEC200P-2는 PROFINET IRT (Isochronous Real Time) 통신에 필요한 PROFINET 모듈을 내장하고 있으며, 함께 제공하는 통신 스택은 해당 모듈을 활용하여, 시간 동기화 및 실시간 통신 등 PROFINET의 다양한 기능을 제공한다. 평가보드에는 ERTEC200P-2 구동에 필요한 SDRAM 및 Flash 메모리가 장착되어 있으며, 디버깅을 위한 JTAG 커넥터를 제공한다.

2.2 PROFINET 통신 스택

PROFINET 통신 스택은 Fig. 1에서와 같이 eCos 운영체제에 해당하는 ECOS 프로젝트와 PROFINET 통신 스택과 어플리케이션 SW가 포함되는 EK_ERTEC 프로젝트로 구분되어 제공된다. ECOS 프로젝트는 빌드시 사전에 설정된 eCos 설정 파일로부터 eCos 소스코드를 생성하고, ERTEC200P-2 구동에 필요한 디바이스 드라이버와 함께 빌드된다. 빌드된 eCos 소스코드들은 라이브러리 파일로 생

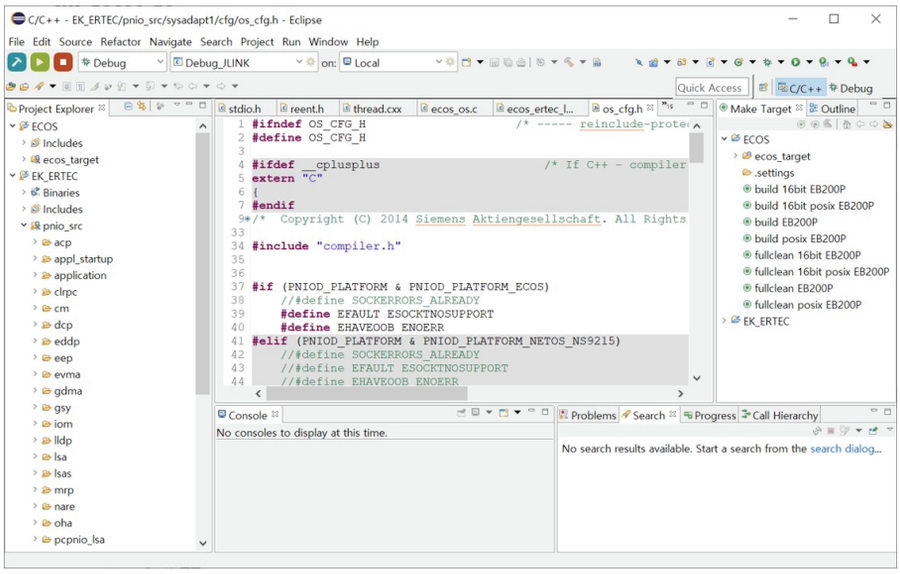

Fig. 1. Development Environment Using Eclipse and gcc

성되어, 관련된 헤더 파일과 함께 EK_ERTEC 프로젝트로 제공된다. EK_ERTEC 프로젝트는 PROFINET 통신 구동에 필요한 통신 스택 및 PROFINET 어플리케이션 개발을 위한 코드를 포함하며, 빌드시 ECOS 프로젝트에 빌드된 라이브러리와 함께 실제 실행 가능한 바이너리를 생성한다. eCos 및 PROFINET 통신 스택은 eclipse v4.5.1 기반에서 arm-non-eabi-gcc v5.3.0를 이용하여 빌드된다. 빌드된 바이너리 파일은 J-Link 등의 에뮬레이터를 이용하여 eclipse 환경에서 다운로드 및 디버깅 가능한 환경을 제공한다.

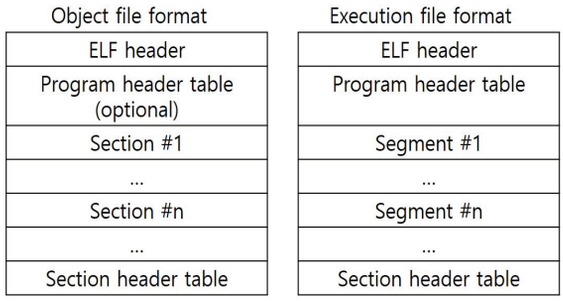

Fig. 2. ELF Format for Object and Executable File

3. PROFINET 통신 SW 포팅 방법

본 장에서는 gcc 기반의 eCos 및 PROFINET 통신 스택을 IAR Studio로 포팅하는 방법을 설명한다.

3.1 IAR 기반 PROFINET 통신 SW 포팅 방안

gcc로 컴파일된 소스코드는 elf 사양에 맞게 오브젝트 파일로 생성된다. elf 포맷은 Fig. 2와 같이 Linker가 사용하는 오브젝트 파일에 대한 포맷과 실행 파일에 대한 포맷을 모두 정의하고 있다. gcc로 컴파일된 eCos 소스코드는 Fig. 2 왼쪽의 elf의 오브젝트 파일 포맷으로 생성되며, IAR 역시 elf 파일 포맷을 지원하기 때문에 gcc에서 컴파일된 eCos의 오브젝트 파일은 IAR 링커에서도 관련 정보를 읽어 링킹할 수 있다.

eCos는 ERTEC200P-2에 대한 디바이스 드라이버, TCP/IP 통신 스택, 멀티 스레드 환경을 제공한다. 이중 디바이스 드라이버 및 TCP/IP 통신 스택은 개발 과정에서 수정이 필요하지 않으며, 멀티 스레드 기능은 개발하고자 어플리케이션에 따라 사용하고자 하는 필요한 스레드의 수와 스레드 별 우선 순위가 상이할 수 있다. 다만, eCos는 컴파일 전에는 스케줄링 알고리즘과 같은 변경할 필요가 없는 항목만 설정하고, 개발 과정에서 필요한 스레드의 생성 및 우선순위 지정 등과 같은 기능은 별도로 제공하는 API (Application Programming Interface)를 활용하여 어플리케이션에서 설정할 수 있다. 이와 같이 PROFINET 개발 과정에서는 eCos의 설정을 변경하고 새로 컴파일해야 하는 경우가 거의 없기 때문에, eCos는 소스코드를 IAR로 포팅하지 않고 gcc로 빌드된 eCos 라이브러리를 재사용하여 IAR 빌드 과정에 함께 링킹될 수 있도록 개발하였다. gcc로 빌드된 eCos 라이브러리를 재사용하므로써 eCos 운영체제 전체를 IAR로 포팅하는 비용을 줄이고, 포팅 과정에서 발생할 수 있는 문제점을 방지할 수 있다.

PROFINET 통신 스택은 모터 드라이브나 I/O 장치에 따라 통신 설정이 변경되어야 하며, 어플리케이션에 따라 통신 설정 및 기능이 변경되어야 한다. 따라서 PROFINET 통신 스택은 어플리케이션에 맞게 변경되고 빌드할 수 있도록 하

여야 하며, 이를 위해 PROFINET 통신 스택은 모든 소스 코드를 IAR 로 포팅하여 어플리케이션 소스코드와 함께 빌드되도록 개발하였다. 이와 같이 eCos와 PROFINET SW를 상이한 방식으로 포팅하여, 개발 과정에서 수정이나 디버깅이 필요하지 않는 코드는 라이브러리 파일을 재사용하여 포팅 비용 및 포팅 시 발생할 수 있는 문제점을 최소화하고, 어플리케이션과 유기적으로 동작하여야 하는 PROFINET 통신 스택은 어플리케이션과 함께 빌드되도록 하여 개발의 효율성을 높였다.

앞서 설명한 바와 같이 이론적으로 elf 표준에 따라 빌드된 eCos 오브젝트 파일은 IAR 링커에서 정상적으로 정보를 인식하고 링킹 가능하다. 하지만 elf는 오브젝트 파일에 대해 정의하고 있으며, 링커의 구현 방법을 정의하고 있지 않기 때문에 IAR 및 gcc의 링커는 상이한 동작 방식을 가지고 있다. 이와 같은 링커의 동작 차이는 본 논문에서 활용하고자 하는 gcc로 빌드된 오브젝트와 IAR 로 빌드된 오브젝트를 동시에 IAR 링커에 의해 링킹되는 경우 몇가지 문제점을 가지고 있다. 첫째 문제는 gcc와 IAR의 Linker Script 작성 방법이 상이하여 모든 항목을 새로 개발하여야 한다. 두 번째 문제는 일부 심볼들의 주소가 비정상적으로 링킹되거나, 정의하지 못하는 문제가 있다. 각각에 대한 세부적인 설명은 아래 세부 절에서 다룬다.

3.2 IAR 기반 Linker Script 작성

Linker Script를 컴퓨터의 메모리를 정의하고, 프로그램의 변수 및 함수들이 포함된 섹션들을 어느 메모리에 저장할지 설정하는 파일이다. Linker Script는 gcc 및 IAR 등 링커에 따라 사용하는 문법이 다르기 때문에 IAR 링커에 맞는 Linker Script 개발이 필요하다. Table 2는 gcc 및 IAR의 Linker Script 파일의 차이를 간략하게 보여준다.

gcc에서는 각 섹션의 시작 주소와 종료 주소를 PC (Program Counter)라고 하는 링커에 의해 할당되는 주소의 현재 값을 '.' 기호를 이용하여 심볼에 저장하고, 이 심볼을 어셈블리나 C 언어에서 extern 하여 사용하는 것이 가능하다. 또한 gcc에서는 특정 섹션이 RAM 저장되어 있다가 수행 전에 DTCM 영역으로 복사되어 동작하는 경우 Table 2와 같이 간단히 '〉' 기호를 이용하여 할당할 수 있다. 반면 IAR 링커는 gcc와 같이 PC('.')를 이용한 섹션(또는 블록)의 시작 및 종료 주소를 가지는 심볼을 정의할 수 없어, SFB (Segment Frame Begin) 및 SFE (Segment Frame End) 와 같은 별도의 어셈블리 명령어를 이용하여 Link Script에서 정의된 블록의 시작 및 종료 주소를 얻어 와야 한다. 또한 특정 섹션이 RAM에 저장되었다가 실제 DTCM으로 복사되어 동작하는 경우에 Table 2에서와 같이 해당 블록의 정의 시 '_init' 으로 끝나는 블록을 추가로 할당하고, '_init' 으로 끝나는 블록이 복사되기 전의 원본 메모리에 할당하여야 한다. 세부적인 스크립트의 작성 방법은 본 논문의 범위를 넘어므로 자세한 내용은 gcc 및 IAR의 Linker 매뉴얼을 참조한다.

Table 2. Example of the Linker Script for gcc and IAR

	Example of the script	Description
gcc	__dtcm_start = .;	Define start addr. of the section
	.datad_tcm : { *(.xdata_d_tcm) __dtcm_end = .;	Define .datad_tcm section Including *.xdata_d_tcm Define end address of the section
	} 〉 dtcm AT 〉 ram	The section is stored in ram memory and then copied to dtcm memory for execution
IAR	define block datad_tcm{ section *.xdata_d_tcm };	Define datad_tcm block Including *.xdata_d_tcm
	define block datad_tcm_init { section *.xdata_d_tcm_init };	The block with postfix (_init) means that it is original block of the copied block to dtcm or RAM
	place in RAM_region { block datad_tcm_init }; place in DTCM_region { block datad_tcm};	Place the defined block to RAM and DTCM region

앞서 설명한 바와 같이 IAR Linker Script 파일에서는 PC를 이용하여 특정 섹션의 시작 및 종료 주소를 심볼로 정의하여 제공하는 기능을 지원하지 않는다. 따라서 gcc에서 Table 2의 __dtcm_start와 같은 심볼이 eCos 및 PROFINET 통신 스택에서 사용된다면 별도의 SW에서 이를 정의해주어야 한다. gcc Linker Script에서 정의하는 주소에 대한 심볼은 MPU의 초기 설정 및 운영체제 부팅을 담당하는 부트로더에서 가장 많이 사용되므로, eCos 내에 포함되어 있는 부트로더 (vectors.S) 파일을 IAR로 포팅하였으며, SFB 및 SFE 명령어를 사용하여, gcc Linker Script에서 정의하고 있는 심볼을 정의하였다. 아래 Table 3은 위 Table 2의 gcc에서 정의한 __dtcm_start 및 __dtcm_end 심볼에 대한 정의 방법과 이를 이용하여 dtcm 데이터 영역을 perform_dma 함수를 호출하여 복사하는 코드이다. Table 3 내에 진하게 강조한 심볼 정의는 __dtcm_start 라는 이름의 라벨에 4byte 공간을 할당하고, 그 위치에 datad_tcm 영역의 시작 주소를 저장한다. 따라서, __dtcm_start 라는 라벨이 가리키는 주소에는 datad_tcm 블록의 시작 주소가

Table 3. Example of the Symbol Definition Using Label

```
  LDR    r5, =___dtcm_start
  LDR    r5, [r5]        /* r5 : start addr. of dtcm */
  LDR    r3, =___dtcm_end
  LDR    r3, [r3]        /* r3 : end addr. of dtcm */
  SUB    r3, r3, r5    /* r3 : size of copy */
  LDR    r6, =0x00 /* incrementing data source
  BL     perform_dma
___dtcm_start: DCD        SFB(datad_tcm)
___dtcm_end:   DCD        SFE(datad_tcm)
```

저장되어 있다. 페이지의 제한에 따라 두 개 심볼을 이용하여 작성 방법을 설명하였고, 실제 구현에서는 flash, ITCM, bss, data 영역 등 12개의 심볼을 추가하였다.

3.3 Map 파일 기반 심볼 주소 생성기

앞서 설명한 SFB 및 SFE 명령을 이용하여 섹션의 시작 주소를 정의하는 기능은 라벨이 심볼의 이름으로 사용되기 때문에, 라벨의 위치(주소)가 심볼의 값이 된다. 따라서, 해당 심볼 값이 가르키는 곳에 원하는 데이터를 저장하여 사용하는 Table 3과 같은 경우는 정상적으로 사용할 수 있으나, 심볼 값 자체를 원하는 상수 값으로 정의해야 하는 경우는 사용할 수 없다. 이를 해결하기 위해 Table 4와 같이 EQU 키워드를 사용하여, 특정 심볼이 특정 상수 값을 가지도록 정의할 수 있다. 정의하는 심볼의 상수 값을 빌드시 생성되는 MAP 파일을 읽어 해당하는 주소를 찾아 텍스트 파일(symbol_def.inc)로 생성하고, 부트로더 소스코드에 include되어 빌드 되도록 설정하였다.

Table 4. Symbol Definition File (symbol_def.inc)

```
        PUBLIC __cached_start
        PUBLIC __cached_end
__cached_start      EQU      0x20005990          //
cached_RAM$$Base
__cached_end  EQU 0x203A7FFF //mmu_tables$$Limit
              ...
        PUBLIC __init_array_start
        PUBLIC __init_array_end
__init_array_start    EQU      0x2020ebf8
__init_array_end      EQU      0x2020ec38

constructors_addr:
  DCD    0x20005990
  DCD    0x20005994
              ...
  DCD    0x200059f0
  DCD    0x20005a10
```

IAR 링커는 C++ 생성자의 시작 주소가 저장되어 있는 Init_array를 할당하지만, Init_array에 호출해야하는 생성자의 주소를 정상적으로 저장하지 못한다. 이를 해결하기 위해 부트로더 내에서 init_array를 이용하여 생성자를 호출하는 로직이 실행되기 전에 init_array에 정상적인 생성자의 주소를 먼저 입력하는 로직을 수행시킨다. Table 5는 앞서 설명한 init_array에 정상적인 생성자의 주소를 저장하는 로직이며, 이를 위해 Table 4에서와 같이 심볼과 생성자의 주소가 저장되어 있는 symbol_def.inc 파일을 생성하는 외부 툴이 필요하다.

앞서 설명한 symbol_def.inc 파일은 부트로더 소스코드에 include 되어 빌드되어야 하기 때문에, 빌드 전에 inc 생성 툴이 호출되어야 한다. inc 생성 툴은 MAP 파일을 입력으로 사용하는데, inc 생성툴이 항상 빌드 전에 호출되어야 하기 때문에 직전의 성공적인 빌드 과정에서 생성된 Map 파일을 입력으로 사용한다. 따라서, inc 파일에 정의된 심볼 및 생성자의 주소는 SW 수정후 빌드하는 경우 다른 값으로 변경될 수 있다. 예는 들면 init_array의 시작 주소가 기존에는 100번지였으나, 새로운 빌드 과정에서는 다른 변수가 추가되어 104번지에 할당될 수 있다. 다만, inc 파일은 init_array가 100번지라고 정의되어 부트로더가 빌드되기 때문에 빌드된 바이너리는 잘못된 주소를 참조하여 정상적으로 동작할 수 없다. 따라서, 바이너리 생성 후 inc 생성 툴에서 inc 파일 내에 정의된 심볼 및 주소 값이 새로 빌드된 MAP 파일의 주소와 동일한지 확인하는 툴이 별도로 필요하다. 두 개의 툴은 모두 Visual Studio 기반의 C언어로 개발하였으며, Fig. 3과 같이 IAR에서 Pre-build 및 Post-build com-

Table 5. Assembly Code for the Correcting Address of the Constructors in init_array Using symbol_def.inc File

```
        LDR r0, =__init_array_start
        LDR r1, NUM_CONSTRUCTOR
        LDR r2, =constructors_addr
loop_copy_const_addr:
        LDR r4, [r2]
        STR r4, [r0]
        ADD r2, r2, #4
        ADD r0, r0, #4
        SUB r1, r1, #1
        CMP r1,#0
        BNE loop_copy_const_addr

#include "symbol_def.inc"
NUM_CON:        DCD        (__init_array_end        -
__init_array_start)/4
```

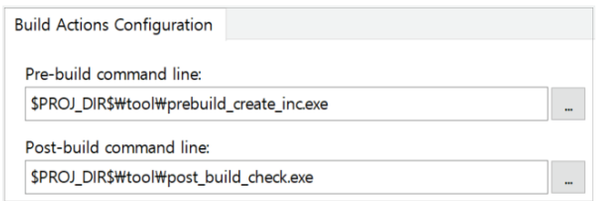

Fig. 3. Pre/Post-build Command Line Configuration

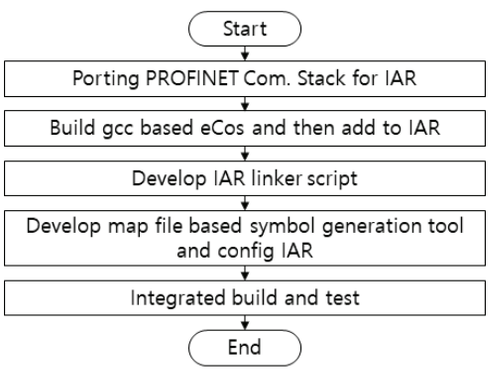

Fig. 4. Flowchart of the Proposed Porting Method

mand line 설정을 통해 빌드 전/후에 자동으로 호출될 수 있도록 설정하였다. prebuild_create_inc.exe 파일은 이전 빌드에서 생성된 MAP파일을 입력으로 하여 Table 3의 symbol_def.inc 파일을 생성하는 역할을 수행하며, Post_build_check. exe 파일은 새로운 빌드에서 생성된 MAP 파일과 빌드에서 사용한 symbol_def.inc 파일의 주소가 동일한지 확인하여, 동일하지 않는 경우 빌드 에러를 발생하는 역할을 수행한다. Fig. 4는 앞서 설명한 제안하는 PROFINET 통신 SW 포팅 방법에 대한 흐름도이다.

4. 실험 환경 및 실험 결과

본 장에서는 제안하는 포팅 방법을 검증하기 위한 검증 환경 및 검증 결과를 설명한다.

4.1 실험 환경

본 논문에서 제안하는 gcc 기반의 eCos 운영체제 및 PROFINET 통신 스택을 IAR 컴파일러 기반으로 포팅하는 방법을 이용하여 IAR Studio V 9.30.1을 이용하여 정상적으로 빌드하였다. 빌드한 바이너리는 Segger사의 J-Link를 이용하여 자체적으로 개발한 PROFINET I/O 임베디드 시스템에 다운로드하여 정상 동작하는 것을 확인하였다.

제안한 포팅 방법을 이용하여 개발한 IAR 기반의 PROFINET 통신 스택이 정상적인 통신 기능을 수행하는 것을 검증하기 위해 Tia Portal을 이용하여 1ms 마다 Digital

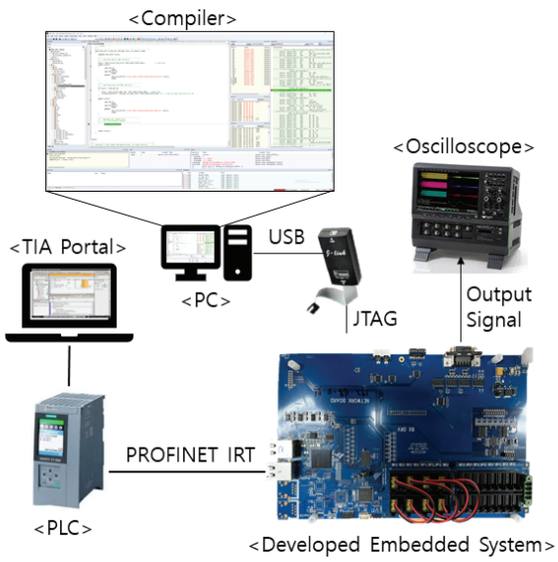

Fig. 5. Experiment Environment

Output을 On/Off 하는 프로그램을 작성하고, 이를 지멘스사의 S7-1515-2 PLC 에 다운로드하였다. PLC와 개발한 임베디드 시스템은 PROFINET IRT로 설정하여 연결하였으며, 통신 싸이클은 1ms로 설정하여 1ms 마다 출력 신호를 on/off 할 수 있도록 설정하였다. 실제 임베디드 시스템이 정상적으로 PROFINET IRT 통신을 수행하여, I/O를 출력할 수 있는지 확인하기 위해 Fig. 5에서와 같이 임베디드 시스템의 디지털 출력 신호를 오실로스코프에 연결하여 동작 여부를 확인하였다.

컴파일 성능을 측정하기 위해서는 동일한 PC에서 동일한 소스 코드를 gcc 및 IAR을 이용하여 빌드하였다. Table 6은 실험 환경 구축하기 위해 사용한 장치의 사양을 보여 준다.

4.2 실험 결과

앞서 실험 환경에서 설명한 바와 같이 PROFINET IRT의 정상 구동 여부를 확인하기 위해 오실로스코프로 측정한 신호는 아래 Fig. 6과 같이 1ms 마다 정상적으로 on/off 되는 것을 확인하였다.

Table 6. Device and SW Specification Used for Experiment

Device / SW		Specification
PC	Compile	I9-11900 (11th gen.), 32GB RAM
	Tia	I7-8700 (8th gen.), 8GB RAM
IAR Studio		EWARM v9.30.1
gcc		gcc v5.3.0 with eclipse v4.5.1
Tia Portal		Tia Portal v17
PLC		S7-1515-2 V2.9
Oscilloscope		Lecroy HD4096

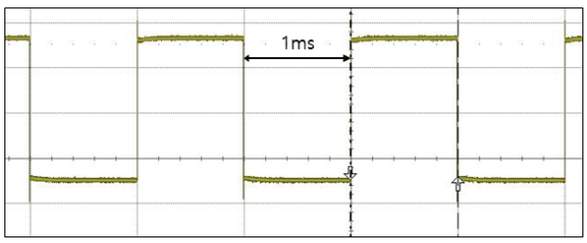

Fig. 6. Measured the PROFINET Output Signal

Table 7. Build Result Using gcc and IAR Studio

	Optimize Opt.	Binary Size	Build time
gcc	-O1	2,483 KB	307s
IAR	Mideum	2,262 KB	68s

또한 컴파일러의 성능을 비교하기 위해 동일한 PC에서 IAR 및 gcc를 이용하여 컴파일한 결과 Table 7과 같이 IAR Studio가 68 s 정도의 빌드 시간이 소요되어 307 s 소요된 gcc 대비 4.5배 정도 빨랐으며, 생성된 바이너리 파일의 크기도 9% 정도 적게 생성되었다. 아래 Table 7에서와 같이 gcc 와 IAR은 최적화 옵션의 설정 방법 및 세부 최적화 내용이 상이하기 때문에 결과를 단순 비교하기는 어렵다. 다만, 일반적으로 최적화 옵션을 강하게 설정하는 경우 빌드 시간이 더 소요되는 점을 고려할 때 빌드 시간이 gcc 보다 4.5배 빠르면서 바이너리 크기가 9% 작기 때문에 IAR이 gcc 대비 빌드 시간 및 바이너리 크기 관점에서 모두 우수하다고 판단된다.

제안한 방법으로 포팅한 IAR 기반의 PROFINET 는 Fig. 6와 같이 Tia Portal을 이용하여 PROFINET IRT로 1ms 마다 Digital Output 신호를 on/off를 반복하도록 PLC에 프로그래밍한 후 지멘스사의 S7-1515-2 PLC에 다운로드하여 실행하여, 오류 없이 정상 구동되는 것을 하였다. 실제 PROFINET IRT 통신과 연동하여 I/O 기능이 정상 동작하는 것을 확인하기 위해 개발한 임베디드 시스템의 Digital Output 신호를 오실로스코프로 측정하였으며, Fig. 6에서와 같이 1ms 마다 정상적으로 신호가 On/off 되는 것을 확인하였다.

5. 결 론

본 논문에서는 gcc 기반으로 개발된 PROFINET SW를 IAR로 포팅하는 방법을 설명한다. 제안하는 방법은 PROFINET SW의 eCos OS는 gcc로 빌드된 라이브러리 파일을 재사용하고 수정이 필요한 PROFINET SW는 IAR로 포팅하여, 효율적으로 PROFINET SW를 개발할 수 있도록 하였다. gcc 및 IAR 링커의 차이로 인해 발생하는 심볼 및 생성자의 주소를 외부 툴을 이용하여 생성하고, 정상적인 주소가 링킹될 수 있도록 관련 코드를 수정하였다. 제안한 방법을 활용하여 개발한 IAR 기반 PROFINET SW는 컴파일 시간 및 바이너리의 크기 모두 gcc 대비 성능이 우수하였으며, PROFINET IRT 통신 기능도 정상 동작하는 것을 확인하였다. 제안하는 방법은 본 논문에서 설명한PROFINET SW 뿐만 아니라 다양한 gcc 기반의 오픈소스를 상용 컴파일로 포팅하는데 도움이 될 것으로 생각한다.

References

[1] M. Felser, "Industrial communication systems and their future challenges: Next-generation ehternet, IIoT, and 5G," *Proceedings of the IEEE*, Vol.107, No.6, pp.944-961, 2019.

[2] T. Moore, Industrial Ethernet Growing but Fieldbus Remains Dominant [Internet], http://www.automation.com/

[3] IEC Std. 61158, Digital data communications for measurement and control - Fieldbus for use in industrial control systems, IEC, 2007.

[4] V. Q. Nguyen, N. V. P. Tran, H. N. Tran, K. M. Le, and J. W. Jeon, "A closed-loop stepper motor drive based on EtherCAT," in *43rd Annual Conference of the IEEE Industrial Electronics Society(IECON)*, pp.3361-3365, 2017.

[5] J. H. Fey, F. Hinrichsen, G. Carstens, and R. Mallwitz, "Development of a modular multilevel converter demonstrator with EtherCAT communication," in *2019 IEEE 13th International Conference on Compatibility, Power Electronics and Power Engineering (CPE-POWERENG)*, pp.1-6, 2019.

[6] C. Kang, Y. Pang, C. Ma, and C. Li, "Development of EtherCAT slave based on multi-core DSP," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp.157-161, 2018.

[7] J. Liu, H. Zhang, X. Guo, and W. Chen, "Design of EtherCAT slave system based on Zynq-7020 chip," in *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp.1916-1920, 2020.

[8] A. L. Dias, G. S. Sestito, A. C. Turcato, and D. Brandao, "Panorama, challenges and opportunities in PROFINET protocol research," in *2018 13th IEEE International Conference on Industry Applications (INDUSCON)*, pp.186-193, 2018.

[9] G. Cheng and X. Wang, "Design of tri-axial motion servo control systems based on EtherCAT," in *2012 IEEE International Conference on Information Science and Technology*, pp.51-54, 2012.

[10] SIEMENS, PROFINET Readme Evaluation Kit ERTEC 200P PN IO V4.7.0 [Internet], https://cache.industry.siemens.com/dl/files/250/109784250/att_1040173/v1/Readme_EvalKit_ERTEC200P_PN_IO_V4.7.0.pdf.

[11] A. J. Massa, Embedded Software Development with eCos, Pearson, 2002.

[12] Linux Foundation, Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification v. 1.2 [Internet], https://refspecs.linuxfoundation.org/elf/elf.pdf.

김 진 호

https://orcid.org/0000-0003-0285-378X
e-mail : kimjh@kyungnam.ac.kr
2007년 성균관대학교 정보통신공학부(학사)
2009년 성균관대학교 정보통신공학부(석사)
2015년 성균관대학교 정보통신공학부(박사)
2015년 ~ 2019년 현대자동차 책임연구원
2019년 ~ 현 재 경남대학교 컴퓨터공학부 조교수
관심분야 : 차량 및 산업용 네트워크, 차량 SW 플랫폼 및 E/E 아키텍처, 실시간 임베디드 시스템, 사물인터넷