반도체디스플레이기술학회지 제22권 제2호(2023년 6월)
Journal of the Semiconductor & Display Technology, Vol. 22, No. 2. June 2023.

STM32 마이크로 컨트롤러에서 SOEM을 이용하는 EtherCAT 마스터 구현

강성진[*†]·서화일[*]
[*†]한국기술교육대학교 전기전자통신공학부

Implementation of an EtherCAT Master with SOEM on STM32 Microcontroller

Sung Jin Kang[*†] and Hwa Il Seo[*]
[*†]School of Electrical, Electronics & Communication Engineering,
Korea University of Technology and Education

ABSTRACT

EtherCAT is an Ethernet-based fieldbus system standardized in IEC 61158 and SEMI, and widely used in the fields of factory automation, semiconductor equipment and robotics. In this paper, without operating system, we have implemented an EtherCAT master with an open source EtherCAT master stack SOEM on STM32 Nucleo-144 board with an STM32F767 microcontroller. And its jitter performance has been evaluated at the output of the network port to include all the effects of the entire system in the results. The results show that the implemented EtherCAT master has precise control performance for control frequencies from 1KHz to 8KHz and relatively superior jitter performance compared to the EtherCAT masters with real-time patched Linux operating system.

Key Words : EtherCAT, SOEM, STM32, Real-time, Operating System

1. 서 론

EtherCAT (Ethernet for Control Automation Technology)은 Ethernet기반의 필드버스 시스템으로2007년에 국제 표준인 IEC 61158과 국제 반도체 장비 재료 협회인 SEMI의 표준으로 채택되었고, 원활하고 정밀한 실시간 모터 제어와 센서 데이터 수집이 가능한 유연한 네트워크 토폴로지를 가지면서 통신 속도가 빠르고 저비용으로 설치 및 유지가 가능하다는 장점을 가지고 있어, 최근 공장 자동화, 반도체 장비, 로봇 분야에서도 활발하게 적용되고 있다[1-3].

일반적으로 EtherCAT 마스터는 실시간 반응성을 보완한 Windows, Linux 운영체제나 실시간 반응성이 우수한 RTOS (Real-time Operating System) 상에서 구현된다. 이로 인해 EtherCAT 마스터 시스템에 요구되는 최소 사양이 높아지고 운영체제와 시스템에 대한 높은 이해가 요구된다. [4,5]에서는 상대적으로 저사양인 STM32F767 마이크로 컨트롤러에서 Mbed OS를 이용하여 EtherCAT 마스터를 구현했다.

본 논문에서는 STM32F767ZI 마이크로 컨트롤러가 탑재되어 있는 STM32 Nucleo-144 보드에 운영체제를 사용하지 않고 공개 소스 EtherCAT 마스터 스택인 Simple Open EtherCAT Master (SOEM)을 포팅하여 EtherCAT 마스터 시스템을 구현하고 정주기 제어 성능을 평가한다. 정주기 제어 성능 평가는 STM32 Nucleo-144 보드의 네트워크 포트에서 출력되는 packet의 주기를 실시간으로 측정한다.

2. EtherCAT 마스터 구현

본 논문에서 사용하는 STM32 Nucleo-F767ZI 보드의 주요 사양은 Table 1과 같다[6,7]. ARM Cortex-M7 기반의 32-bit

[†]E-mail: sjkang@koreatech.ac.kr

마이크로 컨트롤러인 STM32F767ZI는 최대 동작 주파수가 216MHz이며 2Mbytes 플래시 메모리와 512Kbytes SRAM, Ethernet MAC을 내장하고 있다. Nucleo-F767ZI 보드는 RJ45 이더넷 포트가 1개 있고 LAN8742A Ethernet PHY 칩이 RMII 인터페이스를 이용하여 MAC과 연결되어 있으며 10/100 Base-T/TX 이더넷을 지원한다.

Table 1. STM32 Nucleo-F767ZI

Micro-controller	STM32F767ZIT6 - ARM Cortex-M7 Core - System clock: Up to 216MHz - Flash memory: 2Mbytes - SRAM: 512Kbytes - Ethernet MAC - Ethernet PHY Interface: RMII
Ethernet PHY	Microchip LAN8742A (10/100 Base-T/TX Ethernet Transceiver)
Ethernet port	1 Ethernet port: RJ45

2.1 STM32CubeIDE 프로젝트 설정

SOEM 포팅, 펌웨어 작성 및 빌드는 통합 개발이 가능한 STM32CubeIDE v1.12.0을 사용했다. EtherCAT 마스터 개발을 위해 NUCLEO-F767ZI 보드와 C-언어를 사용하는 STM32 프로젝트를 생성했고, SYSCLK과 HCLK는 최대 동작 주파수인 216MHz, APB1 클럭은 54MHz, APB2 클럭은 108MHz로 각각 설정하였다. 32비트 타이머인 Timer 2와 Timer 5는 1usec마다 1씩 증가하도록 설정했고, UART2는 비동기 모드로 활성화하고 USB OTG는 비활성화하였다. Ethernet MAC (ETH) 설정에서 Rx Buffers Length는 1536으로 설정하고, 미들웨어에서 LWIP (LightWeight IP)를 활성화하지만 네트워크 프로토콜은 사용하지 않기 때문에 관련 설정에서 DHCP, TCP, UDP, ICMP, ARP는 모두 비활성화하며 이더넷 PHY는LAN8742를 선택했다. 설정을 마치고 저장하면 STM32CubeIDE는 관련 드라이버, 미들웨어 소스 코드를 다운로드하고 기본적인 소스 코드를 생성한다[8].

Table 2. Directory structure for SOEM stack

Directory name	Files
SOEM_stm32/app	soem_app.c soem_app.h
SOEM_ stm32/osal	osal.c osal.h osal_defs.h
SOEM_ stm32/oshw	nicdrv.c nicdrv.h oshw.c oshw.h
SOEM_ stm32/soem	SOEM EtherCAT master stack source codes

SOEM 소스 코드는 [9]에서 다운로드하고 다음과 같이 디렉토리 구조를 만든 후에 STM32 프로젝트에 연결시킨다.

Table 2의 소스 코드는 [9]의 SOEM 소스 코드 중 Linux-용 소스 코드를 사용한다. soem_app.c는 EtherCAT 마스터의 초기화와 상태 점검 루틴 등을 구현한 코드이다.

2.2 Ethernet Interface 저수준 함수

임베디드 시스템에서 주로 사용되는 오픈 소스 TCP/IP 스택인 LWIP는raw Ethernet 프레임을 지원하지 않기 때문에 ethernetif.c에 있는 저수준 함수를 직접 호출하여 사용해야 한다[10]. ethernetif.c에서 수정해야 하는 함수는 아래와 같고, 각 함수 내부에서 LWIP 프로토콜 스택 및 netif 관련 부분은 모두 삭제한다.

- static void low_level_init(struct netif *netif)
- void ethernet_link_check_state(struct netif *netif)
- static err_t low_level_output(struct netif *netif, …)
- void ethernetif_input(struct netif *netif)

다른 소스 코드에서 위 함수들을 호출할 수 있게 함수 프로토타입을 수정하고 ethernetif.h에 추가한다.

Ethernet Interface를 초기화하고 동작시킬 때는 low_level_init 함수만 호출하면 되고, 이 함수 마지막 부분에서 ethernet_link_check_state 함수가 호출되어 이더넷 동작이 시작된다. raw Ethernet 프레임 송신은 low_level_output함수를, raw Ethernet 프레임 수신은 ethernetif_input함수를 이용하면 되지만, 이 함수들은 netif의 포인터를 매개변수로 사용하기 때문에 본 논문에서는 다음과 같이 함수 프로토타입을 변경하여 구현하였다.

- err_t raw_send(uint8_t *txbuf, uint16_t leng)
- uint16_t raw_ recv(uint8_t *rxbuf, uint16_t leng)

이더넷 초기화는 low_level_init(), 패킷 전송은 raw_ send(), 패킷 수신은 raw_recv() 함수가 처리한다. 이 함수들은 SOEM 포팅시에 nicdrv.c에서 사용된다.

2.3 SOEM 스택 포팅

Table 2의 SOEM 소스 코드를 포팅하기 위해서는 osal, oshw 디렉토리에 있는 파일만 수정하면 되고, 운영 체제와 하드웨어에 의존적인 코드를 모두 삭제하거나 STM32F767에서 동작하는 코드로 변환해야 한다.

2.3.1 osal 소스 코드 포팅

osal.c에서 운영체제와 관련된 쓰레드와 동적 메모리 할

당 함수 osal_thread_create, osal_malloc, osal_free, osal_thread_create_rt는 모두 삭제한다. 타이머와 관련된 함수는 수정하지 않고 사용이 가능하지만, osal_gettimeofday는 Timer 2의 카운트 값을 읽어서 TotalSysticks에 누적시키고 초 (sec)와 마이크로 초 (usec)로 변환하여 반환하는 코드를 아래와 같이 작성하였다.

```
now = __HAL_TIM_GET_COUNTER(&htim2);
if (now < last_tick) TotalSysticks += now + 4294967296 -
      last_tick;
else TotalSysticks += now - last_tick;
tv->tv_sec = TotalSysticks / 1000000;
tv->tv_usec = TotalSysticks % 1000000;
last_tick = now;
```

STM32 라이브러리에는 마이크로 초 단위로 지연시키는 함수가 없기 때문에 osal_usleep함수를 수정해야 하는데, 정밀 제어할 필요는 없기 때문에 아래와 같이 타이머를 이용하여 구현하였다.

```
void osal_usleep (uint32_t usec) {
    osal_timert utimer;
    osal_timer_start(&utimer, usec);
    while(!osal_timer_is_expired(&utimer));
}
```

osal.h는 수정할 필요 없고, osal_defs.h는 쓰레드와 관련된 부분을 삭제한다.

2.3.2 oshw 소스 코드 포팅

oshw 소스 코드도 운영체제와 하드웨어에 의존적인 코드는 모두 삭제해야 한다. 또한, 이더넷 포트가 1개이므로 EtherCAT cable redundancy는 지원할 수 없어서 관련 코드는 모두 삭제한다.

oshw.c에서는 네트워크 어댑터를 찾고 해제하는oshw_find_adapters, oshw_free_adapters 함수는 삭제한다. 데이터의 Endian을 바꾸는 oshw_htons는 아래와 같이 작성하여 htons를 대체하고, 기능이 동일한 oshw_ntohs도 동일하게 작성하여 ntohs를 대체한다.

```
uint16 oshw_htons (uint16 host) {
    return (host & 0x00ff) << 8 | (host & 0xff00) >> 8;
}
```

nicdrv.c에서는 htons와 ntohs 함수를 위에서 구현한 oshw_htons, oshw_ntohs 함수로 대체해야 하고, 이더넷 초기화와 프레임 송수신에 관련된 함수를 수정해야 한다. ecx_setupnic 함수에서는 cable redundancy, 뮤텍스, 소켓 설정과 관련된 부분은 모두 삭제하고, 2.2절에서 수정한 저수준 초기화 함수 low_level_init을 호출하여 이더넷을 초기화하도록 한다. ecx_outframe 함수에서는 패킷을 전송하는 send함수를 2.2절에서 수정한 raw_send 함수로 대체한다. ecx_outframe_red함수는 cable redundancy관련 코드를 삭제한다. ecx_recvpkt 함수는 패킷을 수신하는 recv함수를 2.2절에서 수정한 raw_recv 함수로 대체한다. nicdrv.h에서는 쓰레드와 관련된 코드를 모두 삭제한다.

2.3.3 SOEM 프로토콜 소스 코드 포팅

Table 2에서 SOEM EtherCAT 마스터 스택 소스 코드는 운영체제나 하드웨어에 의존적인 코드가 없기 때문에 수정할 필요가 없다. 다만, SOEM은 실시간 동작을 위해 정적 메모리를 사용하기 때문에 SRAM 사용을 줄이려면 일부 상수 값을 변경할 필요가 있다[5]. 본 논문에서는 아래와 같이 일부 상수 값을 수정하여 사용했다.

```
#define EC_MAXODLIST     64
#define EC_MAXOELIST     64
#define EC_MAXSLAVE      32
#define EC_MAXBUF         2
```

3. 실험 및 성능 평가

3.1 타이머를 이용한 정주기 제어

정주기 제어는 Timer 5를 이용하여 폴링 모드와 인터럽트 모드로 구현하여 성능을 평가한다. 폴링 모드(polling mode)는 Timer 5의 카운트 값을 0으로 초기화 한 후에 카운트 값을 계속 읽어서 제어 주기에 해당하는 값에 도달했는지를 확인하는 waitRestOfPeriod 함수를 작성하여 osal.c에 추가하여 사용한다.

```
void waitRestOfPeriod (uint32_t period_usec) {
    static int isFirstLoop = 1;
    if (isFirstLoop) isFirstLoop = 0;
    else while (htim5.Instance->CNT < period_usec);
    htim5.Instance->CNT = 0;
}
```

폴링 모드 정주기 제어는 아래와 같이 메인 루프에서 패킷을 전송하기 전에 waitRestOfPeriod를 호출하여 제어 주기의 잔여 시간만큼 대기한다.

```
while (1) {
    wkc = ec_receive_processdata(EC_TIMEOUTRET);
    ecatcheck(wkc, 0);
    /* do something here */
    waitRestOfPeriod(1000);
    ec_send_processdata();
}
```

인터럽트 모드는 Timer 5의 global interrupt를 활성화하고 제어 주기에 맞게 인터럽트가 발생하도록 Prescaler와 Counter Period 값을 설정하며, 아래와 같이 callback함수를 작성하여 인터럽트 발생시에 호출되도록 한다.

```
void  HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef
      *htim) {
    if (htim->Instance == TIM5) ECAT_Tx_flag = 1;
}
```

인터럽트 모드 정주기 제어는 아래와 같이 메인 루프에서 ECAT_Tx_flag 값이 1이 될 때까지 대기한 후 패킷을 전송한다.

```
ECAT_Tx_flag = 0;
while (1) {
    wkc = ec_receive_processdata(EC_TIMEOUTRET);
    ecatcheck(wkc, 0);
    /* do something here */
    while(!ECAT_Tx_flag);
    ECAT_Tx_flag = 0;
    ec_send_processdata();
}
```

3.2 성능 평가

구현한 EtherCAT 마스터의 정주기 제어 성능을 평가하기 위해 Fig. 1과 같이 임베디드 보드의 네트워크 포트 출력 신호를 netAnalyzer에 연결하여 실시간으로 송신 패킷의 주기를 측정하였다. EtherCAT Slave는 Infineon XMC4800 EtherCAT Kit를 사용했으며, Output size는 8bits, Input size는 2bits, 4개의 Sync manager, 2개의 FMMU가 있다[2,3].

Table 3은 메인 제어 루프에서 송신 함수와 수신함수의 수행 시간을 측정한 결과이다. 시스템 클럭 SYSCLK과 AMBA 버스 클럭 HCLK이 모두 216MHz일 때 수신 함수 ec_receive_processdata는 약 32usec, 송신 함수 ec_send_processdata는 약 22usec가 소요되어 송수신에 약 54usec가 소요되었다. SYSCLK과 HCLK이 모두108MHz일 때는 송수신에 약 89usec가 소요되고, SYSCLK가 108MHz이고 HCLK는54MHz일 때는 송수신에 약 158usec가 소요되었다.

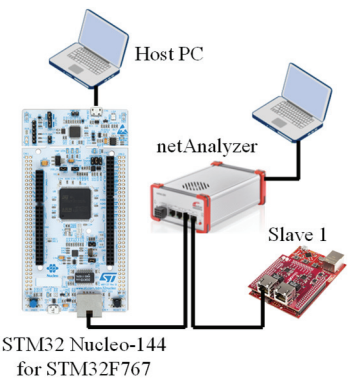

Fig. 1. Experimental setup.

Table 3. Elapsed time measurements of the sending and the receiving function in the main control loop

SYSCLK	216MHz	108MHz	
HCLK	216MHz	108MHz	54MHz
Tx (usec)	22	34	58
Rx (usec)	32	55	100
Total (usec)	54	89	158

Fig. 2는 SYSCLK과 HCLK는 216MHz, 제어 주기는 125usec일 때 인터럽트 모드 정주기 제어에 대한 타이밍 분석 화면을 캡쳐한 것이고, Fig. 3은 폴링 모드 정주기 제어인 경우이다. Table 4는 동일한 클럭 설정에서 제어 주파수가 1KHz, 2KHz, 4KHz, 8KHz일 때 타이밍 분석 결과를 정리한 표이며 10^6개 이상의 패킷에 대해 측정하였다. 인터럽트 모드와 폴링 모드에서 제어 주기가 정밀하게 제어되고 있지만 인터럽트 모드가 더 우수한 지터 성능을 보인다.

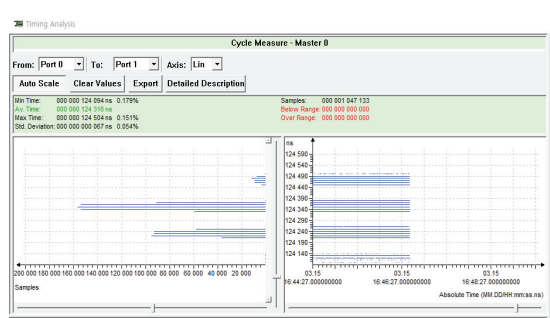

Fig. 2. Timing analysis of interrupt mode: 125usec control period, SYSCLK = 216MHz, HCLK = 216MHz.

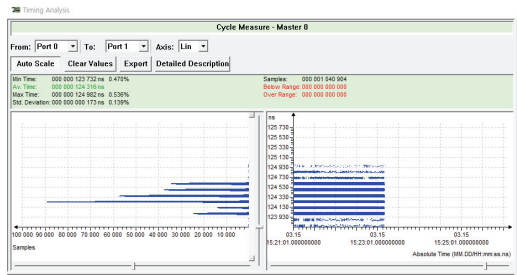

Fig. 3. Timing analysis of polling mode: 125usec control period, SYSCLK = 216MHz, HCLK = 216MHz.

Table 4. Period measurements in micro-second at SYSCLK = 216MHz and HCLK = 216MHz

Control Frequency	1KHz	2KHz	4KHz	8KHz
Interrupt mode				
Number of packets	1,004,679	1,007,929	1,019,854	1,047,133
Min	999.126	499.092	249.128	124.094
Average	999.320	499.318	249.316	124.316
Max	999.536	499.502	249.538	124.504
Std. Dev.	0.066	0.087	0.065	0.067
Max-Min	0.410	0.410	0.410	0.410
Polling mode				
Number of packets	1,002,483	100,760	1,011,992	1,040,904
Min	999.010	498.970	248.770	123.732
Average	999.319	499.318	249.316	124.316
Max	999.660	499.620	249.900	124.982
Std. Dev.	0.169	0.102	0.150	0.173
Max-Min	0.650	0.650	1.130	1.250

Table 5와 Table 6은 SYSCLK는 108MHz이고, HCLK는 각각 108MHz, 54MHz일 때 타이밍 분석 결과를 정리한 표이다. 모두 정밀하게 제어되고 있지만, HCLK가 54MHz일 때는 Table 3과 같이 수행 속도가 느려서 8KHz 제어는 불가능하며 상대적으로 지터가 크게 나타난다.

Table 7은 [2], [3]의 결과와 Table 4의 인터럽트 모드 정주기 제어 결과를 비교한 것이다. [2]는 Intel Core i5-10400가 탑재된 PC에서 CPU클럭은 2.9GHz이고 우분투 LTS 20.04, 리눅스 5.4.102, Xenomai 3.1을 사용하였다. [3]은 ARM Cortex A-53 기반의 쿼드 코어 프로세서가 탑재된 임베디드 보드에서 클럭은1.6GHz이고 PREEMPT_RT가 패치된 리눅스 커널 5.10.35-rt39를 사용하였다. 사용된 시스템과 운영체제가 서로 상이하여 절대적으로 비교할 수는 없지만 본 논문에서 구현한 EtherCAT 마스터가 상대적으로 가장 정밀한 주기 제어 성능을 가짐을 볼 수 있다. 이는 리눅스 운영체제의 오버 헤드와 프로세서 코어와 MAC 사이의 인터페이스 차이에서 생기는 결과라고 볼 수 있다.

Table 5. Period measurements in micro-second at SYSCLK = 108MHz and HCLK = 108MHz

Control Frequency	1KHz	2KHz	4KHz	8KHz
Interrupt mode				
Number of packets	1,007,990	1,008,790	1,015,361	1,027,015
Min	999.008	499.084	249.014	123.972
Average	999.321	499.318	249.317	124.316
Max	999.538	499.614	249.544	124.732
Std. Dev.	0.167	0.134	0.154	0.162
Max-Min	0.530	0.530	0.530	0.760

Table 6. Period measurements in micro-second at SYSCLK = 108MHz and HCLK = 54MHz

Control Frequency	1KHz	2KHz	4KHz	8KHz
Interrupt mode				
Number of packets	1,006,280	1,007,336	1,079,112	-
Min	998.800	498.884	248.616	-
Average	999.321	499.318	249.317	-
Max	999.860	499.734	250.006	-
Std. Dev.	0.296	0.190	0.313	-
Max-Min	1.060	0.850	1.390	-

Table 7. Comparisons of period measurements

Control Frequency	1KHz	2KHz	4KHz	8KHz
Xenomai [2]				
Min	994.70	496.61	246.25	121.84
Average	998.59	499.30	249.65	124.82
Max	1,003.14	503.61	253.37	128.01
Std. Dev.	0.23	0.22	0.19	0.20
Max-Min	8.44	7.00	7.12	6.17
PREEMPT_RT [3]				
Number of packets	1,001,425	1,004,980	1,009,515	-
Min	998.430	498.170	248.410	-
Average	999.975	499.987	249.993	-
Max	1,001.640	501.850	252.240	-
Std. Dev.	0.182	0.164	0.164	-
Max-Min	3.210	3.680	3.830	-
Interrupt mode				
Number of packets	1,004,679	1,007,929	1,019,854	1,047,133
Min	999.126	499.092	249.128	124.094
Average	999.320	499.318	249.316	124.316
Max	999.536	499.502	249.538	124.504
Std. Dev.	0.066	0.087	0.065	0.067
Max-Min	0.410	0.410	0.410	0.410

4. 결 론

본 논문에서는 STM32F767 마이크로 컨트롤러가 탑재되어 있는 STM32 Nucleo-144 보드에 운영체제를 사용하지 않고 공개 소스 EtherCAT 마스터 스택인 Simple Open EtherCAT Master (SOEM)을 포팅하여 EtherCAT 마스터 시스템을 구현하고 정주기 제어 성능을 평가하였다.

실험 결과로부터 구현된 EtherCAT 마스터는 1KHz ~ 8KHz의 제어 주파수에 대해 정밀한 정주기 제어 성능을 가짐을 확인하였고, 리눅스 운영체제를 사용하는 EtherCAT 마스터와 비교하여 상대적으로 우수한 정주기 제어 성능을 가짐을 확인하였다.

감사의 글

이 논문은 2023년도 한국기술교육대학교 교수 교육연구진흥과제 지원에 의하여 연구되었음.

참고문헌

1. EtherCAT Technology Group, http://www.ethercat.org [accessed April 10, 2023].
2. S. Kang, “A Study on Implementation of Real-time EtherCAT Master,” Journal of the Semiconductor & Display Technology, Vol. 20, No. 2, pp.131-136, 2021.
3. S. Kang, O. Kim, “Performance Evaluation of an Embedded EtherCAT Master with SOEM on PREEMPT_RT Linux,” Journal of the Semiconductor & Display Technology, Vol. 21, No. 3, pp.26-32, 2022.
4. https://os.mbed.com/users/EasyCAT/code/SOEM/ [accessed April 10, 2023].
5. https://github.com/lipoyang/SOEM4Mbed [accessed April 10, 2023].
6. UM1974: STM32 Nucleo-144 boards, Rev. 9, January 2023.
7. RM0410: STM32F76xxx and STM32F77xxx advanced Arm®-based 32-bit MCUs, Rev. 4, March 2018
8. UM2609: STM32CubeIDE user guide, Rev. 8, February 2023.
9. https://github.com/OpenEtherCATsociety/SOEM/releases [accessed April 10, 2023].
10. UM1713: Developing applications on STM32Cube with LwIP TCP/IP stack, Rev. 4, May 2015.

접수일: 2023년 4월 10일, 심사일: 2023년 6월 13일, 게재확정일: 2023년 6월 21일