Journal of Institute of Control, Robotics and Systems (2016) 22(8):671-677
http://dx.doi.org/10.5302/J.ICROS.2016.16.0089 ISSN:1976-5622 eISSN:2233-4335

FPGA를 이용한 CAN 통신 IP 설계 및 구현

Design and Implementation of CAN IP using FPGA

손 예 슬, 박 정 근*, 강 태 삼
(Yeseul Son[1], Jungkeun Park[1,*], and Taesam Kang[1])
[1]Department of Aerospace Information Engineering, Konkuk University

Abstract: A Controller Area Network (CAN) is a serial communication protocol that is highly reliable and efficient in many aspects, such as wiring cost and space, system flexibility, and network maintenance. Therefore, it is chosen for the communication protocol between a single chip controller based on Field Programmable Gate Array (FPGA) and peripheral devices. In this paper, the design and implementation of CAN IP, which is written in VHSIC Hardware Description Language (VHDL), is presented. The implemented CAN IP is based on the CAN 2.0A specification. The CAN IP consists of three processes: clock generator, bit timing, and bit streaming. The clock generator process generates a time quantum clock. The bit timing process does synchronization, receives bits from the Rx port, and transmits bits to the Tx port. The bit streaming process generates a bit stream, which is made from a message received from a micro controller subsystem, receives a bit stream from the bit timing process, and handles errors depending on the state of the CAN node and CAN message fields. The implemented CAN IP is synthesized and downloaded into SmartFusion FPGA. Simulations using ModelSim and chip test results show that the implemented CAN IP conforms to the CAN 2.0A specification.

Keywords: CAN, FPGA , VHDL, serial communication

I. 서론

최근 차량 시스템에 사용되는 마이크로 컨트롤러와 전자장비의 개수가 크게 증가하고 있다. 고급 사양의 자동차의 경우 50개 이상의 전자제어유닛(ECU)이 사용되기도 한다[1]. 전자장비의 개수가 증가함에 따라 배선이 복잡해지고 무게가 증가하였으며, 주고 받는 정보의 양도 증가하였다. 이러한 다수의 ECU간 통신을 위해서는 점대점(Point-to-Point) 방식의 연결이 아닌 버스 시스템이 요구된다[2]. Controller Area Network(CAN)은 Bosch사에서 차량 네트워크용으로 개발한 직렬 통신 프로토콜이다[3]. 다중 마스터, 브로드캐스팅 방식을 사용하여 모든 노드가 메시지를 수신하고 식별자에 따라 필요한 메시지를 사용한다. 또한 모든 노드에서 에러를 확인하고 결함이 있는 노드의 연결을 끊는 등 신뢰성이 매우 높은 프로토콜이다. 이러한 특징 때문에 CAN은 최근 차량 시스템 뿐만 아니라 항공우주 시스템, 산업용 자동화 기기, 제어기 등 여러 분야에서 채택되어 사용되고 있다.

한편, 마이크로컨트롤러를 내장한 저가의 FPGA 칩들이 개발되면서 단일 칩으로 복잡한 제어를 수행할 수 있게 되었다[4]. 이러한 단일 칩 제어기 상에서 외부기기와의 연결을 위해 CAN이 요구되는 경우가 많다. 이 경우 IP 모듈로 CAN 통신을 구현하게 되면 별도의 칩을 사용하는 단가를 줄일 수 있다. 그러나 CAN 프로토콜은 메시지의 길이가 길고, 우선순위 기반 메시지 전송과 고신뢰 메시지 전송을 위해 복잡한 동기화 과정과 에러 감지 및 처리과정이 필요하여 구현이 어렵다. CAN 프로토콜을 모두 지원하는 CAN IP 모듈은 대부분 상용 코드로 사용이 어려우며[5] 일부 구현된 VHDL 코드들은 기능이 제한적이다[6-8].

본 논문에서는 FPGA로 구현된 단일칩 제어기와 외부 기기간의 통신을 위해 VHDL을 이용하여 CAN 컨트롤러 모듈을 구현하고 이를 FPGA에 탑재하여 구동한다. 본 논문에서 구현한 CAN IP는 CAN 2.0A 통신 규격을 따르며 다른 IP 모듈과 합성하여 재사용이 가능하다. 이전 논문[9]에서는 CAN의 송수신 기능만을 구현하여 시뮬레이션을 통해 기능을 확인하였다. 본 논문에서는 이를 확장하여 CAN 2.0A 표준에 명시된 모든 기능을 구현 완료하였다. 본 논문에서는 3개의 VHDL 프로세스로 CAN IP를 설계, 구현하고 이를 합성하여 FPGA 칩에 탑재한다. 그리고 시뮬레이션과 상용 CAN 디바이스와의 통신을 통하여 구현된 CAN IP가 표준에 맞게 동작하는지를 확인하였다.

II. CAN 2.0A 표준

1. 버스

CAN은 직렬 통신 프로토콜로 CAN 2.0A는 최대 1Mbit/s의 전송 속도를 지원한다[3]. CAN은 두 개의 전선을 버스로 사용하고, 버스 레벨은 두 전선 간의 전압 차를 이용하여 하이(high)와 로우(low)가 아닌 우성(dominant)과 열성(recessive)으로 구분된다. Wired-AND 방식을 사용한다면 '0'은 우성, '1'은 열성으로 표현된다. 버스에서 우성과 열성이 충돌하면 버스 레벨은 우성으로 나타나게 된다. 이러한 특성에 의해 여러 노드에서 동시에 메시지가 전송될 때 식별자의 우선순위에 따라 버스 중재가 가능하며, 송신 노드에서 ACK (acknowledgement) 비트가 우성 비트가 되는지 확인하여 메시지 전송오류를 체크할 수 있다.

* Corresponding Author
Manuscript received April 18, 2016 / revised June 2, 2016 / accepted June 24, 2016
손예슬, 박정근, 강태삼: 건국대학교 항공우주정보시스템공학부
(yeseul0323@naver.com/parkjk@konkuk.ac.kr/tskang@konkuk.ac.kr)
※ 본 논문은 2016 제 31회 제어로봇시스템학회 학술대회에 초안이 발표되었음.

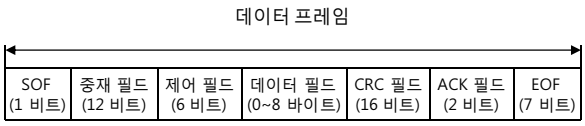

그림 1. 데이터프레임의 필드 구성.
Fig. 1. Structure of fields in a data frame.

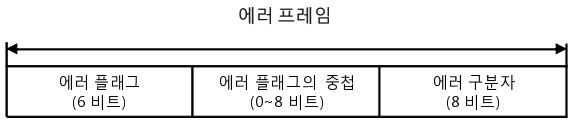

그림 2. 에러프레임의 필드 구성.
Fig. 2. Structure of fields in an error frame.

2. 메시지

CAN은 메시지 단위로 데이터를 전송한다. 메시지는 기능에 따라 5 가지 프레임으로 분류된다. 각 프레임은 다른 노드로 데이터를 전송하기 위한 데이터 프레임, 다른 노드에 데이터 전송을 요청하기 위한 원격 프레임, 에러가 발생했음을 알리기 위한 에러 프레임, 오버로드 상황이 발생했음을 알리기 위한 오버로드 프레임, 메시지 간의 거리를 조종하기 위한 인터프레임 스페이스 이다.

데이터 프레임은 그림 1과 같이 7개의 필드로 구성되어 있다. 프레임 시작(SOF: Start-Of-Frame)은 메시지의 시작을 알리는 필드로 1 비트의 우성 비트이다. 중재필드는 메시지가 여러 노드에서 동시에 전송되었을 때 우선순위를 결정하기 위한 필드이다. 11 비트의 식별자와 1 비트의 원격 전송요청 비트(RTR bit)로 구성되어 있다. 원격 전송요청 비트가 우성이면 메시지는 데이터 프레임이고, 열성이면 원격프레임이다. 제어필드는 두 개의 우성 비트인 예약 비트와 4 비트의 데이터 길이 코드(DLC: Data Length Code)로 구성되어 있다. 데이터 길이 코드는 0~8 바이트까지의 데이터 길이를 이진수로 표현한 것이다. 데이터필드는 전송되는 데이터이다. CRC 필드는 오류 검출을 위한 필드로, 15 비트의 CRC 시퀀스와 1 비트의 열성 비트인 CRC 구분자로 구성되어 있다. ACK 필드는 1 비트의 ACK 슬롯과 1 비트의 열성 비트인 ACK 구분자로 구성되어있다. 송신 노드는 ACK 슬롯(slot)에서 열성 비트를 송신하고, 모든 수신 노드는 수신된 메시지에 오류가 없을 시 우성 비트를 송신한다. 송신 노드는 우성 비트를 수신하여 오류가 없음을 확인할 수 있다. 프레임 종료(EOF: End-of-Frame)는 프레임이 끝났음을 알리는 필드로 7 비트의 열성 비트이다.

원격 프레임은 다른 노드에 데이터 전송을 요청하기 위한 프레임이다. 원격 프레임은 데이터 프레임에서 데이터 필드를 제외한 모든 필드를 가지고 있다.

데이터 프레임과 원격 프레임은 에러 발생을 줄이고 클락 동기화를 위해 SOF부터 CRC 필드까지 비트 스터핑(bit stuffing) 방식을 사용한다. 같은 레벨의 비트가 5개 이상 연속되면 반대 레벨의 비트를 삽입해 준다. 따라서 메시지의 길이는 같은 바이트 수의 데이터를 가지더라도 다를 수 있다.

에러 프레임은 에러가 발생했을 시 다음 비트부터 즉시 전송이 시작된다. 그림 2와 같이 에러 프레임은 에러 플래그와 에러 구분자 두 개의 필드로 구성되어 있다. 에러 플래그는 6 비트이며 노드가 에러 액티브(error active)일 시 우성 비트, 노드가 에러 패시브(error passive)일 시 열성 비트로 구성된다. 에러 구분자는 8개의 열성 비트로 이루어져 있다. 모든 노드가 동일한 시점에 에러 플래그 전송을 시작하지 않으므로 에러 컨디션을 종료하기 위해서 노드는 에러 구분자의 첫 번째 비트를 전송하고 버스에 열성 비트가 수신되길 기다린다. 열성 비트가 수신되면 에러 구분자의 두 번째 비트부터 전송을 시작한다.

오버로드 프레임은 다음 두 가지 상황에서 전송된다. 수신 노드가 내부적으로 다음 데이터 프레임 또는 원격프레임을 수신할 준비가 되지 않았을 때, 인터프레임 스페이스 전송 중 우성 비트가 수신되었을 때, 오버로드 프레임의 전송은 액티브 에러 프레임과 동일하게 동작한다.

인터프레임 스페이스는 데이터프레임 또는 원격프레임이 전송된 후 전송되는 프레임으로, 프레임 간의 간격을 주기 위해 전송된다. 3개의 열성 비트로 이루어져 있다.

3. 메시지 중재

CAN은 우선순위에 기반한 메시지 중재를 지원한다. 두 개 이상의 노드에서 동시에 메시지를 전송할 경우 식별자의 우선순위가 높은 메시지가 우선 전송된다. 우선순위가 낮은 메시지를 전송하던 노드는 전송을 중단하고 수신 노드가 된다. 수신이 완료되면 메시지를 처음부터 다시 전송한다. 식별자의 상위 비트가 우성 비트인 경우가 우선순위가 높다. 버스에서 열성 비트와 우성 비트가 충돌하면 우성 비트가 샘플링되기 때문에 식별자를 전송하는 중 비트 에러가 발생하면 다른 노드에서 우선순위가 높은 메시지를 전송하고 있는 것으로 간주한다.

4. 비트 타이밍

CAN에서 1 비트를 송수신할 때 비트 타임은 클락 동기화와 지연 보상, 버스 레벨 샘플링을 위해 그림 3과 같이 4개의 세그먼트로 이루어져 있다. 클락 동기화를 위한 동기화 세그먼트(synchronization segment), 전송 지연보상을 위한 지연시간 세그먼트(propagation time segment), 데이터를 샘플링하고 클락 재동기화를 위한 위상 버퍼 세그먼트1(phase buffer segment1)과 위상 버퍼 세그먼트2(phase buffer segment2)가 있다. 동기화 세그먼트의 길이는 1 time quantum(TQ)이고, 나머지 각 세그먼트는 여러 개의 TQ로 이루어져 있으며, TQ의 길이와 개수는 계산에 의해 결정된다. 데이터 샘플링은 위상 버퍼 세그먼트1과 위상 버퍼 세그먼트2 사이에서 이루어진다. 모든 하강 에지에서 동기화가 이루어지며, 프레임 시작

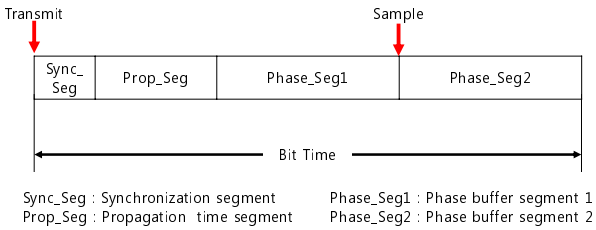

그림 3. Bit time의 구성.
Fig. 3. Structure of a bit time.

(SOF) 수신 시에는 경동기화(hard synchronization), 그 이외 하강 에지에서는 재동기화(resynchronization)가 이루어진다. 하강 에지의 위치에 따라 에지가 늦을 경우(late edge)에는 위상 버퍼 세그먼트 1의 길이를 늘이고, 에지가 빠를 경우(early edge)에는 위상 버퍼 세그먼트 2의 길이를 줄인다. 동기화 점프 간격(synchronization jump width)을 설정하여 조정 한계를 정해 둔다.

III. CAN IP 설계 및 구현

1. CAN IP 설계 환경

본 논문에서는 CAN IP 구현을 위해 Microsemi사의 SmartFusion A2F 500M3G 484FBG를 사용한다. SmartFusion 칩은 ARM사의 Cortex-M3 마이크로프로세서와 500,000개의 게이트와 11,520개의 D-flip-flop을 가진 FPGA가 내장되어 있다[10]. 본 논문에서는 VHDL로 CAN 통신을 담당하는 IP를 작성하여 FPGA에 합성한다. 합성 툴로는 Microsemi사에서 제공하는 Libero SoC를 사용하였고 ARM Cortex 프로그래밍은 SoftConsole을 사용하였다.

SmartFusion의 마이크로프로세서에서는 응용 프로그램이 수행되며 CAN IP의 레지스터에 전송할 메시지를 쓰고, CAN IP가 레지스터에 쓴 수신된 메시지 읽어온다. 마이크로프로세서와 FPGA의 통신을 위해 Microsemi사에서 제공하는 AHB-Lite Bus IP를 사용하였다.

2. CAN IP 구조

본 논문에서는 CAN 2.0A 표준을 만족하는 CAN IP를 설계 및 구현한다. CAN IP는 그림 4와 같이 클럭 생성 프로세스, 비트 타이밍 프로세스, 비트 스트리밍 프로세스로 구성되고 각각의 프로세스의 역할은 다음과 같다.

- 클럭 생성 프로세스: 시스템 클럭을 카운트하여 단위 클럭(time quantum clock)을 만든다. 단위 클럭은 비트 타이밍 프로세스에서 비트 타이밍에 맞게 비트를 읽고 쓸 때 사용된다.

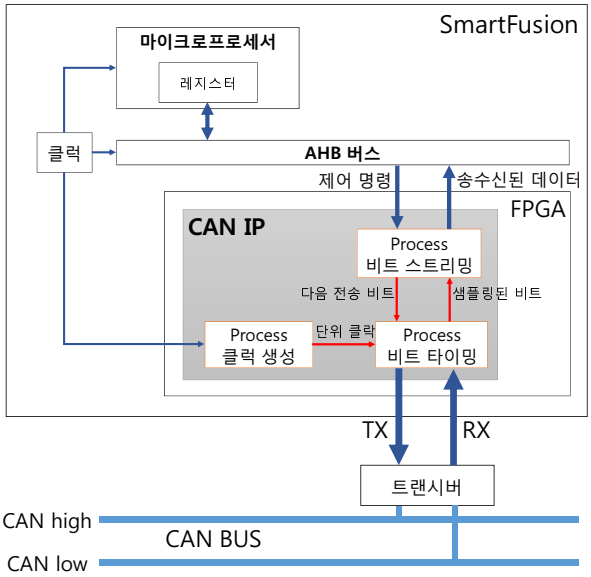

그림 4. CAN IP의 구조.

Fig. 4. Structure of the CAN IP.

```
Entity CAN_IP is
Port(Reset, Clk : IN std_logic;
     TX : OUT std_logic;
     RX : IN std_logic);
End CAN_IP;

Architecture CAN_IP of CAN_IP is
Begin
     Process: Clock_Generator(Reset, Clk)
     Process: Bit_Timing(Reset, Tq)
     Process: Bit_Streaming(Reset, Process_Bit)
End CAN_IP
```

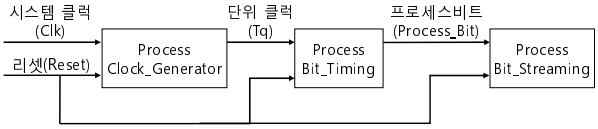

그림 5. CAN IP 프로세스의 감응리스트와 생성 감응 시그널.

Fig. 5. Sensitive list and generated sensitive signal of process in CAN IP.

- 비트 타이밍 프로세스: CAN의 비트 타이밍에 맞게 동기화를 수행하고, 데이터를 Tx 포트로 출력하고 Rx 포트로부터 버스 레벨을 읽는다.
- 비트 스트리밍 프로세스: 수신된 비트를 처리하고 다음에 전송될 비트 레벨을 결정하여 비트 타이밍 프로세스에 전달한다. CRC계산과 스터핑(stuffing), 디스터핑(de-stuffing), 에러 검출과 처리도 이 프로세서에서 진행된다.

클럭 생성 프로세스에서 생성된 단위 클럭 시그널은 비트 타이밍 프로세스의 동작 클럭(sensitivity clock)이 되고, 비트 타이밍 프로세스에서 생성된 프로세스 비트는 비트 스티밍 프로세스의 동작 클럭이 된다. 모든 프로세스는 감응 시그널의 상승 에지에서 동작한다. 리셋은 마이크로프로세서에서 들어오는 리셋을 공통적으로 사용하며 액티브 로우(active low)로 동작한다. 각 프로세스의 감응 리스트와 그 프로세스에서 생성되어 다른 프로세스의 감응리스트로 사용되는 시그널의 관계, 프로세스의 역할은 그림 5에 정리되어 있다.

그림 6은 CAN IP의 상태 머신(state machine)을 보여준다. RECEIVING 상태에서 메시지를 수신하고 TRANSMITTING 상태에서 메시지를 송신한다. IDLE은 버스에 어떤 메시지도

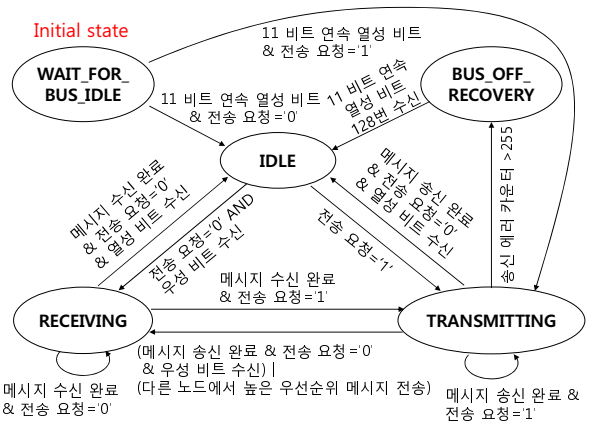

그림 6. CAN IP의 상태 머신.

Fig. 6. State machine of the CAN IP.

전송되지 않는 상태이다. WAIT_FOR_BUS_IDLE은 노드가 처음 동작할 때 초기 상태로, 버스가 IDLE 상태가 되길 기다린다. BUS_OFF_RECOVERY 상태에서는 노드에 결함이 있어 차단되었을 때 복구를 위한 작업을 수행한다. 본 상태 머신을 기반으로 CAN IP의 메시지 송수신과 에러 처리구현을 다음 장에 설명한다.

3. 비트 송수신과 클럭 동기화

CAN에서 1비트를 송수신 하기 위한 비트 타임은 4개의 세그먼트로 구성된다. CAN은 시스템 클럭을 받아 생성한 단위 클럭(time quantum clock)을 기본 클럭으로 사용한다. 단위 클럭의 길이는 계산에 의해 결정되며, 4개의 세그먼트는 여러 개의 단위 클럭을 소모한다. 클럭 생성 프로세스에서는 시스템 클럭을 카운트하여 단위 클럭을 생성한다. 이 단위 클럭을 받아 비트 타이밍 프로세스에서 비트 타이밍 동작이 이루어진다. 비트 타이밍 프로세스는 단위 클럭의 상승 에지에서 버스 레벨을 샘플링하고 현재 세그먼트의 상태와 비트 내에서의 위치를 파악하고 동기화와 버스 입출력을 수행한다. 버스 레벨의 하강 에지를 감지하면 노드와 세그먼트 상태에 따라 경동기화 또는 재동기화를 수행한다. 위상 버퍼 세그먼트 1이 끝나고 위상 버퍼 세그먼트 2가 시작되는 지점에서 버스 레벨을 저장하고, 위상 버퍼 세그먼트 2가 끝나고 동기화 세그먼트가 시작되는 위치에서 저장되어 있는 비트를 TX 핀에 출력한다. 버스 레벨을 저장한 다음 단위 클럭에서 비트 스트리밍 프로세스의 감응 시그널이 되는 프로세스 비트를 1 TQ 동안 하이로 만들어 주어 비트 스트리밍 프로세스를 트리거한다. 노드의 상태와 전송할 비트 레벨은 비트 스트리밍 프로세스에서 결정된다. 수신된 비트를 처리하고 다음 전송할 비트를 결정할 때까지 걸리는 시간은 1 비트를 처리하는 데 걸리는 시간이다. 이를 정보 처리 시간(IPT: information processing time)이라 하며 이는 샘플 포인트(sample point)의 상승 에지에서 프로세스 비트의 하강 에지까지 2 TQ의 시간 이다. 그림 7은 샘플 포인트와 프로세스 비트, 전송 포인트의 관계를 보여준다.

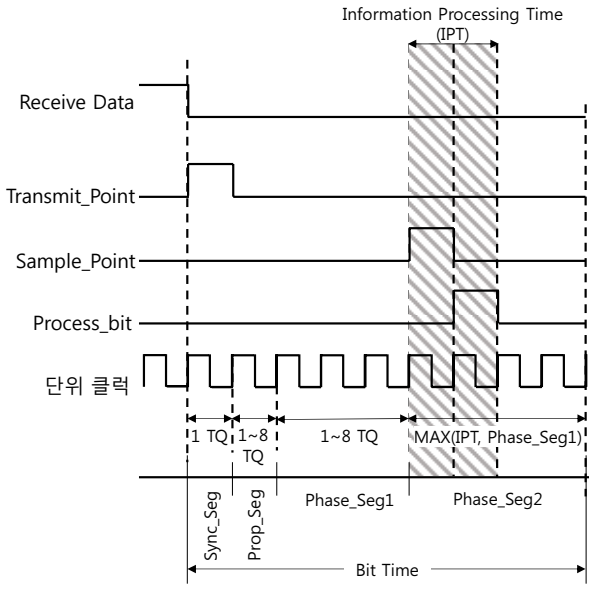

그림 7. SAMPLE_POINT와 PROCESS_BIT, TRANSMIT POINT의 관계.

Fig. 7. Relation between SAMPLE_POINT, PROCESS_BIT and TRANSMIT_POINT.

4. 메시지 수신

메시지 수신은 비트 타이밍 프로세스와 비트 스트리밍 프로세스에 의해 이루어진다. 비트 타이밍 프로세스에서 비트를 샘플링하고 저장한 후, 비트 스트리밍 프로세스의 감응 시그널인 프로세스 비트를 하이로 만들어 준다. 그림 8은 비트 스트리밍 프로세스의 수신 흐름도를 보여준다. 비트 스트리밍 프로세스에서는 현재 노드의 상태와 수신된 비트가 어떤 필드의 몇 번째 비트인지 추적한다. 현재 수신된 비트가 SOF 일 경우 메시지의 수신을 시작한다. 현재 수신된 비트가 스터핑 비트인지 판단하고 CRC를 계산한 후, 메시지 파싱을 수행한다. 식별자와 데이터 길이 코드, 데이터인 경우 버퍼에 이를 저장한다. 구분자와 CRC 필드인 경우 에러를 검사하고, 에러 필드인 경우 현재 수신된 비트의 유효성을 판단한다. 다음 필드가 수신답신 슬롯이고 에러가 없다면 출력할 비트에 우성 비트를 저장한다. 에러 발생 시, 수신 에러 카운터는 증가하고 필드는 에러 플래그로 변경되며 포지션은 1로 변경된다. 메시지 수신이 완료되지 않았다면 필드와 포지션을 다음 필드와 포지션으로 갱신하고 메시지 수신이 완료되었으면 버퍼에 저장된 식별자와 데이터 길이, 데이터를 레지스터에 저장한다. 메시지 수신이 완료되거나 에러 플래그의 전송이 끝난 경우 수신된 비트와 전송 요청 상태를 확인하여 노드의 상태를 변경시킨다.

5. 메시지 송신

메시지 송신은 수신과 마찬가지로 비트 타이밍 프로세스와 비트 스트리밍 프로세스에 의해 이루어진다. 그림 9는 비

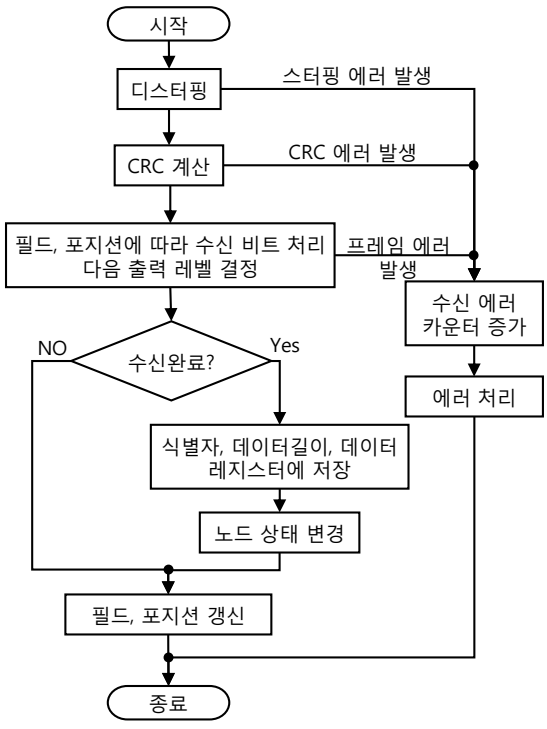

그림 8. 비트 스트리밍 프로세스의 수신 흐름도.

Fig. 8. Flow chart of bit streaming process during receiving.

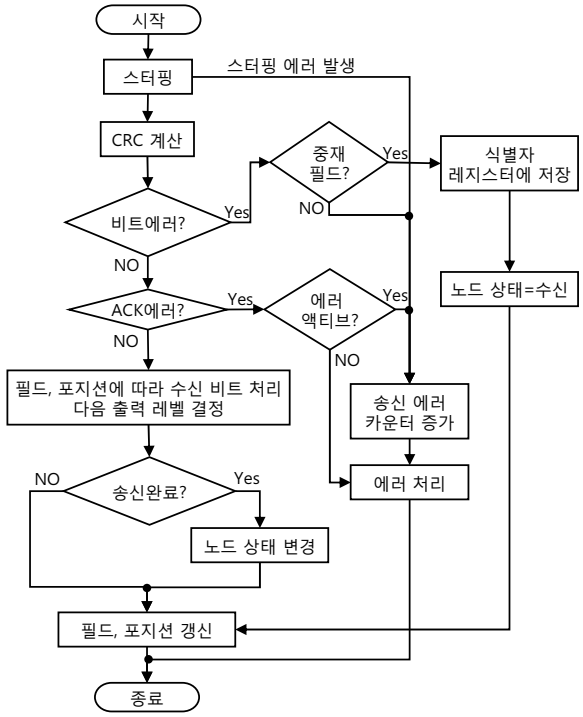

그림 9. 비트 스트리밍 프로세스의 송신 흐름도.
Fig. 9. Flow chart of bit streaming process during transmitting.

트 스트리밍 프로세스의 송신 흐름도를 보여준다. 비트 타이밍 프로세스에서 샘플링한 비트를 다음 비트가 스터핑 비트인지 판단하고, CRC를 계산한다. 노드에서 버스로 출력한 비트 레벨과 샘플링한 비트 레벨이 일치하는지 확인하여 비트 에러를 검사한다. 이 때, 중재 필드에서 비트 에러가 발생하면 다른 노드에서 우선순위가 높은 메시지를 송신하고 있는 것으로 판단하고 현재까지 송신한 식별자를 수신 버퍼에 저장하고 노드의 상태를 수신으로 바꾸어 다음 비트부터 메시지 수신을 시작한다. 필드와 포지션으로 추적한 현재 비트의 위치로 형식 에러(form error)와 ACK 에러를 검사하고 다음 출력할 비트 레벨을 결정한다. 에러 발생 시, 수신 에러 카운터를 증가시키고 그에 따라 노드 에러 상태가 에러 액티브, 에러 패시브, 버스 오프인지 판단한다. 노드가 에러 패시브일 경우 수신답신 에러는 에러 카운터를 증가시키지 않는다. 노드가 에러 액티브 또는 패시브일 경우 필드를 에러 플래그로 변경하고 포지션을 1로 두어 다음 비트부터 에러 플래그를 전송하도록 한다. 노드의 에러 상태가 버스 오프라면 노드 상태를 버스 오프 회복으로 변경한다

6. 에러 처리

노드가 수신 또는 송신 상태일 때 에러가 발생하면 버스 오프를 제외하고 노드의 상태는 변경되지 않은 상태에서 에러 플래그를 전송한다. 에러 발생 시, 노드가 수신 상태이면 수신 에러 카운터를 증가시키고 송신 상태이면 송신 에러 카운터를 증가시킨다. 송신, 수신 에러 카운터가 모두 128 미만이면 노드는 에러 액티브 이다. 송신 또는 수신 에러 카운터 중 하나가 128 이상이면 노드는 에러 패시브이다. 송신 에러 카운터가 256 이상이면 노드에 결함이 있다고 판단되고 버스에서 차단되어 복구 과정에 들어간다.

에러 플래그를 6 비트 전송한 후 노드는 에러 구분자의 첫 번째 비트를 전송하고 열성 비트가 수신되길 기다린다. 이 때, 노드는 추가적으로 8비트까지 에러 플래그가 중첩되는 것을 허용한다. 열성 비트가 수신되면 나머지 에러 구분자를 모두 전송한다. 액티브 에러 플래그를 송신하는 중 비트 에러가 발생하면 새로운 에러 플래그를 전송한다. 에러 구분자를 전송하는 중 우성 비트가 발견되면 오버로드 컨디션이 발생한 것으로, 오버로드 플래그를 전송한다. 에러 구분자까지 전송을 마치면 노드는 전송 요청이 있으면 송신 상태가 되고, 전송 요청이 없으면 아이들 상태로 변경된다.

노드가 차단된 경우 노드의 상태는 버스 오프 회복 상태가 되고 모든 메시지의 송수신을 중단한다. 연속된 11개의 열성 비트가 128번 수신되면 결함이 없다고 판단되어 에러 카운터를 초기화 하고 노드를 다시 복구한다.

IV. 실험 및 평가

1. 실험 환경

실험은 두 가지 단계로 진행되었다. 먼저 시뮬레이션을 통해 동작과 타이밍을 확인한 후, SmartFusion 칩에 다운로드하여 칩 테스트를 수행하였다. 시뮬레이션은 논리 회로의 동작과 성능을 실험할 수 있는 ModelSim을 사용하여 프로세스에 사용되는 모든 시그널들의 동작과 타이밍을 확인하였다. 칩 테스트는 상용 CAN 디바이스와의 호환성을 확인하기 위해 CAN 라이브러리가 내장되어 있는 mbed [11]와 통신 시켰다.

실험은 메시지 수신과 송신, 메시지 중재, 에러 처리 네 가지 경우에 대해 진행하였다. 실험 전송 속도는 50 kHz이고, 샘플 포인트는 80%에 위치하며 1 TQ의 길이는 2 μs, 전체 TQ의 개수는 10 개, 지연시간 세그먼트의 길이는 5 TQ, 위상 버퍼 세그먼트1과 위상 버퍼 세그먼트2의 길이는 각각 2 TQ로 지정하였다. 송수신과 에러 테스트 위한 메시지의 식별자는 11 비트 0b00000001010, 데이터는 1 바이트(byte) 0b10101010이고, 중재 테스트를 위한 mbed의 식별자는 0b00000001010, SmartFusion의 식별자는 0b00000001011이다. 송신 에러와 수신 에러 카운터는 칩 테스트로 에러 생성이 어려워 시뮬레이션을 통해 확인하였다.

2. 실험 결과

메시지 수신 실험에서는 SmartFusion이 수신노드, mbed가 전송노드로 실험하였다. 실험 결과는 그림 11과 같다.

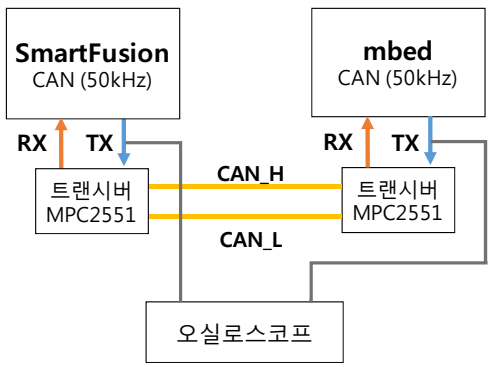

그림 10. 실험 환경 구성.
Fig. 10. Test environment.

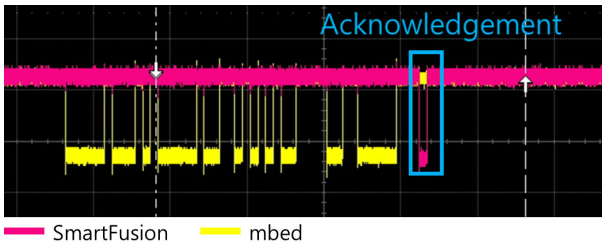

그림 11. 수신 실험 오실로스코프 측정 화면.
Fig. 11. Measurement of receiving test.

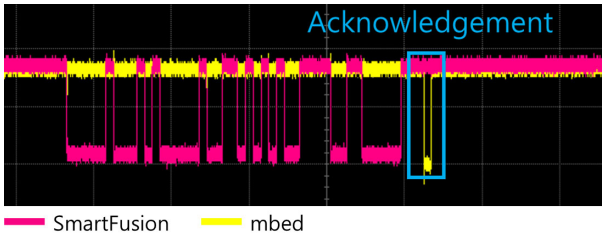

그림 12. 송신 실험 오실로스코프 측정 화면.
Fig. 12. Measurement of transmitting test.

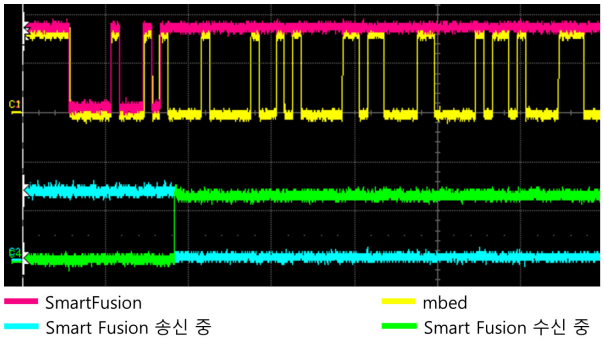

그림 13. 중재 실험 오실로스코프 측정 화면.
Fig. 13. Measurement of arbitration test.

SmartFusion이 메시지를 에러 없이 수신하여 ACK 슬롯의 위치에서 'dominant' 비트를 전송하였다.

송신 실험에서는 SmartFusion이 전송노드, mbed가 수신노드로 실험하였다. 실험 결과는 그림 12와 같다. 그림 12에서 mbed가 메시지를 에러 없이 수신하여 ACK 슬롯의 위치에서 'dominant' 비트를 전송하였다. mbed에 수신된 메시지는 식별자 0b1010, 데이터 0b10101010으로 전송된 메시지와 일치한다.

우선순위에 대한 실험은 SmartFusion에서 전송하는 메시지 식별자가 0x0b이고, mbed에서 전송하는 메시지 식별자가 0x0a로 SmartFusion의 우선순위가 낮은 경우에 대해 진행되었다. 그림 13에서 SmartFusion이 중재 필드에서 비트 에러를 발견하고 전송을 중단하고 수신 노드가 되는 것을 확인할 수 있다.

에러 처리 실험에서는 SmartFusion이 전송 노드이고 다른 CAN 노드와 연결되지 않아 ACK를 받지 못해 에러가 발생하도록 하였다. 메시지를 송신하다 ACK 슬롯에서 'dominant' 비트를 감지하지 못해 ACK 에러가 발생한다. Transmit_Error_counter가 128 미만이고 Receive_Error_counter가 128 미만이면 노드는 Error Active이다. 에러 발생 시 노드는 Active

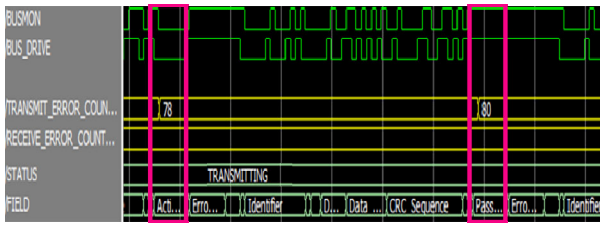

그림 14. 에러 카운터에 따른 에러플래그 전송 시뮬레이션.
Fig. 14. Simulation of transmitting error flag following the error counters.

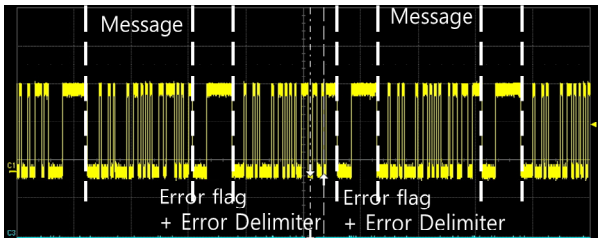

그림 15. Active error 오실로스코프 측정 화면.
Fig. 15. Measurement of active error test.

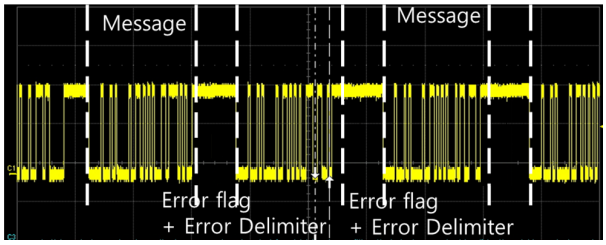

그림 16. Passive error 오실로스코프 측정 화면.
Fig. 16. Measurement of passive error test.

에러 플래그와 에러 구분자를 송신한다. 이후 다시 메시지를 전송한다. Transmit_Error_counter가 128 이상이거나 Receive_Error_counter가 128 이상이면 노드는 Error Passive가 되고 Passive 에러 플래그와 에러 구분자를 전송한다.

그림 14에서 TRANSMIT_ERROR_COUNTER가 0x80(십진수 128) 미만일 때는 Active 에러 플래그를 전송하고 0x80 이상일 때는 Passive 에러 플래그를 전송함을 확인 할 수 있다.

그림 15와 그림 16은 에러 발생 시, 버스 레벨을 오실로스코프로 측정한 것이다. 그림 15는 노드가 Error Active일 때의 파형이고, 그림 16은 노드가 Error Passive일 때의 파형이다.

Transmit_Error_counter가 256 이상이 되면 노드에 결함이 있다고 판단되어 노드는 BUS_OFF 상태가 된다. 송신이 차단된 상태에서 수신만 가능하며, 11개의 연속된 열성 비트(recessive bit)가 수신되는 것을 카운트한다. 128번 수신되면 노드에 결함이 없다고 판단되어 노드를 다시 복구시킨다. 노드는 IDLE 상태로 돌아가며, Error active가 된다. TRANSMIT_ERROR_COUNTER와 RECEIVE_ERROR_COUNTER는 모두 0으로 초기화 된다.

그림 17와 18는 결함 노드 차단 및 복구에 대한 시뮬레이션 결과이다. TRANSMIT_ERROR_COUNTER가 0x100(십진수 256) 이상이 되면 Error Passive 상태에서 BUS_OFF_RECOVERY 상태로 바뀌게 된다. 송신을 멈추고 POSITION

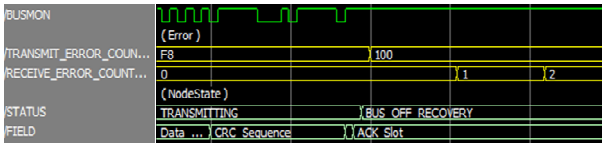

그림 17. 에러 카운터에 따른 노드 차단 시뮬레이션.
Fig. 17. Simulation of bus-off following the error counters.

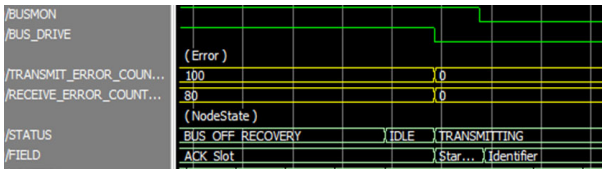

그림 18. 에러 카운터에 따른 차단 노드 복구 시뮬레이션.
Fig. 18. Simulation of bus-off recovery following the error counters.

과 RECEIVE_ERROR_COUNTER를 이용하여 11개의 연속된 열성 비트의 수신을 카운트한다. 그림 17에서 TRANSMIT_ERROR_COUNTER가 0x100 이상이 되어 노드가 BUS_OFF_RECOVERY 상태가 되고, 그림 18에서는 노드가 128(0x80)번의 연속된 11개의 열성 비트를 수신한 후 IDLE 상태로 복귀한다.

V. 결론

본 논문에서는 FPGA로 설계된 단일칩 제어기에 사용하기 위하여 FPGA를 이용한 CAN IP을 설계하였다. CAN IP는 클럭 생성, 비트 타이밍, 비트 스트리밍 3개의 프로세서로 구성하였고 Bosch에서 배포한 CAN 2.0 표준의 요구사항에 맞추어 비트 타이밍, 메시지 프레이밍, 메시지 송수신, 유효성 체크, 에러 확인 및 에러 처리, 결함 노드 차단을 구현하였다. 구현된 CAN IP가 표준에 맞게 동작하는지 확인하기 위해 ModelSim을 이용하여 시뮬레이션을 수행하고 SmartFusion 칩의 FPGA에 구현된 CAN IP를 다운로드하여 상용 CAN 디바이스와의 호환성을 검증하였다.

REFERENCES

[1] K. S Yi and J. Y Lee, "Vehicle dynamics control applications to automobiles: Survey and some new trends," *Journal of Institute of Control, Robotics and Systems*, vol. 20, no. 3, pp. 298-312, Mar. 2014.
[2] M. Khanapurkar, P. Bajaj, and S. D. H. Wandhare, "Design approach for VHDL and FPGA implementation of automotive black box using CAN protocol," *International Journal of Computer Science and Network Security*, vol. 8, no. 9, Sep. 2008.
[3] Controller Area Network, Robert Bosch Gmbh, CAN Specification version 2.0, 1991.
[4] M. Yoon, J. Park, and T. Kang, "Single-chip controller design for piezoelectric actuators using FPGA," *Journal of Institute of Control, Robotics and Systems*, vol. 22, no. 7, pp. 513-518, Jul. 2016.
[5] VHDL Reference CAN, Robert Bosch Gmbh., User's Manual, 1999.
[6] B. E. Sreeram Krishnamoorthy, "Desing of an ASIC chip for a Controller Area Network (CAN) protocol controller," Graduate Faculty of Texas Tech University, Aug. 2006.
[7] T. Hulawale, N. Koul, and S. Gupta, "FPGA based CAN protocol controller," *International Journal of Advanced Technology in Engineering and Science*, vol. no. 3, no. 01, Jan. 2015.
[8] M. Khanapurkar, J. Y. Hande, and P. Bajaj, "Approach for VHDL and FPGA implementationh of communication controller of FlexRay controller," *Journal of Information Hiding and Multimedia Signal Processing Ubiquitous International*, vol. 1, no. 04, Oct. 2010.
[9] Y. Son, J. Park, T. Kang, and J. H. Gu, "Implementation of CAN IP using FPGA," *Proceedings of Institute of Control, Robotics and Systems*, Seoul, Korea, pp. 229-230, Mar. 2016.
[10] http://www.microsemi.com/
[11] https://developer.mbed.org/

손 예 슬

2015년 8월 건국대학교 항공우주공학과 졸업. 2015년 9월~현재 동 대학원 석사과정 재학 중. 관심분야는 항공제어, 항공전자, FPGA 응용.

박 정 근

1997년 서울대학교 전기공학부 졸업. 1999년 동 대학원 석사. 2004년 서울대학교 전기·컴퓨터공학부 박사. 2008년 3월~현재 건국대학교 항공우주공학과 교수. 관심분야는 임베디드 실시간 운영체제, 실시간 시스템.

강 태 삼

1986년 서울대학교 제어계측공학과 졸업. 1988년 동 대학원 석사. 1992년 동 대학원 박사. 2001년 9월~현재 건국대학교 항공우주공학과 교수. 관심분야는 항공제어, 소형비행체 및 관성센서 지능소자 활용 강인제어 이론 및 응용.