FPGA 기반의 임베디드 프로세서 시스템을 이용한 CAN 통신 인터페이스 구현

An Implementation of CAN Communication Interface using the Embedded Processor System based on FPGA

구태묵*, 박영석**
Tae-Mook Koo, Young-Seak Park

요 약

최근 전자제어 차량을 비롯한 각종 산업용 임베디드 시스템은 분산형 다중 마이크로 컨트롤러 시스템으로 진화하고 있다. 이에 따라 제어의 효율성이 큰 객체지향형 시스템 구축이 용이하고, 통신의 높은 안정성과 신뢰성이 보장되는 표준적 CAN(Controller Area Network) 통신 규약이 필요하게 되었다. 기존의 범용 프로세서를 이용한 CAN 통신 인터페이스는 하드웨어 아키텍처가 고정되어 있기 때문에 다양한 응용에 적용함에 있어 유연성이 결여되는 등의 많은 한계를 가진다. 본 논문에서는 FPGA 기반 CAN 통신 인터페이스 시스템을 설계 구현하고, 기존의 AT90CAN128 컨트롤러와의 통신 성능을 모니터링 하여 시스템의 기능과 성능을 검증하였다. 본 연구의 CAN 인터페이스 시스템은 IFI_Nios_II_Advanced_CAN IP 코어와 NIOS II 소프트 코어 프로세서를 사용하여 설계 되었다.

이에 따라 개발된 CAN 통신 인터페이스는 다양한 FPGA 기반 응용 시스템 개발에 재사용 될 수 있고, 저비용, 소형화 그리고 저전력화를 달성할 수 있다.

Abstract

Recently, various industrial embedded systems including vehicles controlled electronically are evolving to distributed multi-micro controller system. Accordingly, there is a need for standard CAN(Controller Area Network) protocol that ensures high stability and reliability of communication and is simple to construct object-oriented system with high control efficiency. CAN communication interface used general-purpose processor doesn't have many limitations in various application development because of fixed hardware architecture.

This paper design and implement a CAN communication interface system based on FPGA. It is verified function and performance of system through monitoring communication with existing AT90CAN128 controller.

Implemented CAN communication interface can be reused in development of application systems based on FPGA. And it provides low-cost, small-size and low-power design advantages

Keywords : Embedded Processor system, CAN(Controller Area Network)

Ⅰ. 서론

전자기술의 발달과 온-칩(On-Chip) 네트워킹 프로토콜의 출현으로 자동차뿐만 아니라 각종 산업용 임베디드 시스템이 중앙 집중형 제어 시스템에서 분산형 다중 MCU(Micro Controller Unit) 시스템으로 진화하고 있다.

기존의 All-in-One 시스템은 구조 및 통합 제어 시스템이 복잡하고 다양해지면서 외부 배선이 복잡해지고, 많은 양의 데이터 처리 요구에 대한 MCU의 부하가 증가하여 시스템의 확장 및 유지 보수에 어려움이 있었다. 이를 위해 1986년 독일의 로버트 보쉬(BOSCH)에 의해 최초로 개발되고, 125Kbps까지의 활용을 위해 ISO-11519로, 최고 속도 1Mbps까지의 활용을 위해 ISO-19898로 표준화된 CAN(Controller Area Network) 통신 규약이 필요하게 되었다 [12][18].

CAN은 고속의 통신 인터페이스를 제공하고, 식별자(Identifier)를 이용하여 노드간의 충돌 방지와 전송 중재 기능을 가지고 있어 실시간 계측 제어 시스템에 적합하다. 또한, 외부의 전자기적인 방해요인에 강인한 장점을 가지고 있다. 이러한 특징 때문에 차량과 산업용 임베디드 시스템, 심지어 커피 기계에서부터 신장 결석 쇄석기에 이르기까지 점차 응용의 폭이 넓어지고 있다[12]~[14].

*코위버(주) **경남대학교 정보통신공학과
투고 일자 : 2009. 7. 15 수정완료일자 : 2009. 11. 5
게재확정일자 : 2010. 1. 29
※본 연구는 2009년도 경남대학교 학술연구장려금 지원으로 이루어졌음.

이렇듯 시스템의 성능이 좋아지고 사용자 편의 장치들이 늘어남에 따라 시스템의 전기·전자적 구조 및 통합 제어 시스템이 점점 복잡해지고 이를 제어하기 위한 소프트웨어의 양과 복잡도도 함께 증가하게 되었다. 이러한 현실에 직면한 개발 업체들은 시스템 제어 기능의 효과적인 개발을 위해 표준화된 통합 소프트웨어 플랫폼의 필요성을 절감하게 되었다.

기존의 범용 CAN 컨트롤러를 이용한 CAN 통신 인터페이스는 하드웨어 아키텍처가 고정되어 있기 때문에 유연성이 전혀 없어 자원을 너무 적거나 너무 많이 가지게 되고, 다양한 응용에 있어서 많은 한계를 가진다. 그러나 반도체 공정 기술이 발전함에 따라 수천만, 수억 개의 게이트를 가진 FPGA(Field Programmable Gate Array) 칩이 개발됨에 따라 하나의 칩 안에 시스템 전체의 모든 기능을 구현 할 수 있게 되었다. 이는 시스템의 통합 소형화와 폭 넓은 응용 및 적응성을 향상 시킬 수 있고, 저 전력화를 달성하여 시스템의 휴대성을 증가시킬 수 있다. 또한, 최근 FPGA 가격이 크게 떨어짐에 따라 생산 비용 절감의 효과도 기대할 수 있다. 특히 Altera사의 Nios II 소프트코어 임베디드 프로세서[3]~[11]는 사용이 간편하고 신뢰할 수 있는 고성능을 제공하기 때문에 설계 커뮤니티 및 산업현장에서는 표준 FPGA 프로세서로 널리 이용되고 있다.

본 논문에서는 Nios II 소프트코어 프로세서를 사용하여 FPGA 기반의 CAN 통신 인터페이스 시스템을 설계 구현하고, 기존의 AT90CAN128 컨트롤러[23]와의 통신 성능을 모니터링 하여 시스템의 기능과 성능을 검증하였다.

II장에서는 CAN을 소개하고 III장에서는 CAN 인터페이스 설계를, IV장에서는 실험 및 결과, V장에서는 결론을 요약한다.

II. Controller Area Network 소개

CAN(Controller Area Network)은 1988년 Bosch와 Intel에서 자동차 산업 분야에 적용하기 위해 고안되었으며, 물리 계층과 데이터 링크 계층만으로 구성되어 있는 시리얼 네트워크 통신 방식이다.

현재 ISO 11898 및 11591-1 표준안으로 지정되어 있다. 일반적으로 사용되는 CAN 버스는 마이크로 컨트롤러 사이에서 통신망을 형성하며, 2가닥의 꼬임 쌍선으로 연결하여 짧은 메시지를 사용하는 고속 응용 시스템에 적당하다. 외부로부터의 잡음에 강인성을 가져 통신 에러율을 최소화하여 높은 신뢰성을 가지고 있다. 통신 속도는 실시간 제어가 가능한 1Mbps(ISO 11898 규격)의 고속 통신을 제공하며, 심각한 잡음 환경에 적합하도록 에러 검출, 에러 보정 기능 및 재전송의 기능이 있다.

2.1 CAN의 데이터 링크 계층

2.1.1 Bus Arbitration의 원리

CAN 버스에서 두 개 이상의 노드들이 동시에 데이터를 전송할 때, 한 데이터가 우선순위를 갖는 방법이 필요하다. 이것은 'Non-Destructive Bit-wise Arbitration'을 사용하여 달성된다. CAN 버스의 각 노드는 고유한 메시지 식별자(Identifier)를 가진다. 이는 11비트(표준형) 또는 29비트(확장형)로 구성된다. 이와 같이 CAN 버스의 접근을 위한 중재는 식별자의 각 비트들을 충돌 시켜서 우선순위를 결정하는 충돌 탐지 기능을 가진 Carrier Sense Multiple Access with Collision Detection(CSMA/CD) 메커니즘을 기반으로 한다[12].

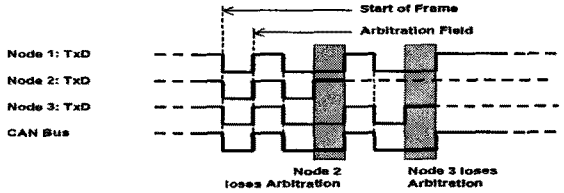


그림 1. Bus Arbitration의 원리

Fig. 1.Principle of Bus arbitration

CAN 버스는 'Wired-AND' 로직에 의해서 디지털 '0'이 디지털 '1'에 우선한다. 따라서 CAN 버스상에서 '0'의 비트 값을 전송하는 최초의 노드가 버스에 대한 가장 높은 우선순위를 갖게 되고, 성공적으로 자신의 메시지를 전송하게 된다.

2.1.2 CAN 프레임 포맷

CAN의 메시지 프레임은 Data Frame, Remote Frame, Error Frame, Overload Frame의 네 종류로 나누어지며, 각각의 기능은 아래 <표-1>과 같다.

표 1. CAN 메시지 프레임의 종류

Table 1. Type of CAN message frames

종 류	기 능
Data Frame	전송 노드에서 수신 노드로 데이터를 운반
Remote Frame	원격 노드에 데이터 프레임의 전송을 요구
Error Frame	버스 에러 검출 시 에러를 검출한 노드에서 전송
Overload Frame	내부적인 과부하 상태로 data frame이나 remote frame을 지연 시킬 필요가 있거나, 에러 조건이 있을 때 발생

2.1.3 오류 취급(Error Handling)

CAN 프로토콜은 매우 효율적인 에러 검출 메커니즘을 가지고 있다. 전송에 포함된 모든 노드들은 어떠한 메시지라도 일단 접수하면 에러 처리 과정에 동참한다. 그림 2는 CAN에서 검출되는 에러의 유형을 보인다. 모든 노드는 모든 프레임에 대해 에러를 체크를 하고, 시스템상의 어떤 노드에서 에러가 검출되면 즉시, 송신 측으로 신호를 보낸다.

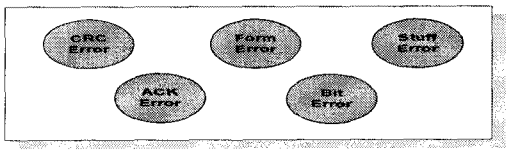


그림 2. 에러의 유형

Fig. 2. Type of errors

2.2 CAN의 물리 계층

2.2.1 CAN 버스 동기화

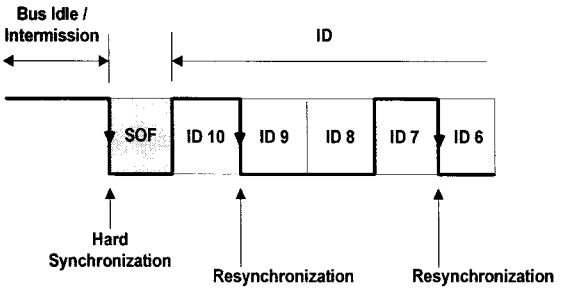


그림 3. CAN 버스 동기화
Fig. 3. CAN Bus synchronization

그림 3에서와 같이 CAN 버스 동기화는 프레임의 시작 부분에서의 'Hard Synchronization'과 프레임 내부에서의 'Re-synchronization'의 두 가지로 나누어 설명할 수 있다. 버스상의 모든 노드는 첫 번째 falling edge를 가지는 각 메시지 프레임의 시작 부분인 SOF 구간에서 동기화되며, 이를 'Hard Synchronization'이라 한다. 마지막 비트까지 정확한 샘플링을 하기 위해서 CAN 노드들은 전체 프레임 동안에 재동기화를 해야 할 필요가 있다. 이는 각각 'recessive'에서 'dominant' 에지의 falling edge에서 이루어진다[12][18].

CAN은 자신의 오실레이터로 클럭 되는 다수의 노드들로 구성되기 때문에 위상 이동(Phase Shift)이 여러 노드들에서 일어날 수 있다. 서로 다른 CAN 노드는 CAN 프레임을 수신하는 동안에 오실레이터 허용 오차와 위상 이동을 보상하기 위해 재동기화를 수행한다.

2.2.2 물리적 전송 매체

그림 4는 CAN 버스상의 5V의 differential한 전압 레벨을 보인다. CAN 버스 라인은 'recessive'(Logic 1)와 'dominant'(Logic 0)의 CAN_High와 CAN_Low의 신호로 나누어진다. 따라서 두 비트 상태를 전송할 수 있는 두 가닥의 꼬임 쌍선을 전송선로로 사용한다. 그리고 꼬임 쌍선의 양 끝단은 임피던스 매칭 문제를 해결하기 위하여 120 Ohm의 종단 저항을 달아주어야 한다[12].

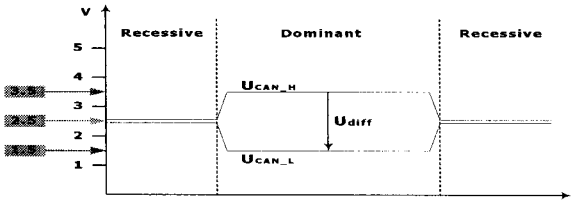


그림 4. CAN 버스의 전압 레벨(CAN_H, CAN_L)
Fig. 4. Voltage level of CAN Bus

CAN 버스상의 신호는 Electromagnetic Interference(EMI) 발생 시 두 개의 버스 라인에서 동시에 영향을 받아 변하더라도 두 신호의 전압 차는 항상 동일하므로 동일한 값의 신호를 전송할 수 있는 장점이 있다. 이렇듯 CAN은 전자파나 외부 잡음과 같은 전자기적인 방해요인에 강인한 특성을 가진다.

III. CAN 통신 인터페이스 설계

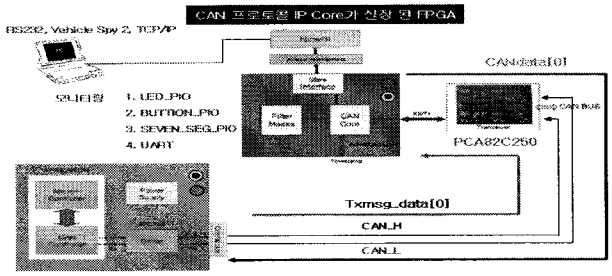


그림 5. CAN 통신 인터페이스 시스템 구성도
Fig. 5. Structure of CAN communication interface

그림 5는 본 논문에서 구현하고자하는 CAN 통신 인터페이스 시스템의 전체 구성도를 나타낸다. Nios II 개발 보드에서 CAN 통신 인터페이스를 위한 주변 인터페이스들은 Quartus II(SOPC Builder)를 이용하여 구성한다. 그리고 멀티 마스터(Multi-Master), 브로드캐스팅(Broadcasting)으로 전송 방식을 설정하고, CAN IP 코어에서 송수신되는 CAN 메시지의 구성과 이벤트 등은 Nios II 프로세서를 이용하여 구현한다.

인터페이스를 위한 외부의 CAN 노드는 Atmel사의 AT90CAN128 컨트롤러 통합 칩을 이용하여 CAN 테스트 보드를 제작하였다. 그리고 CAN IP 코어가 실장 된 FPGA 기반의 임베디드 프로세서 시스템과 상호 통신되는 데이터를 LED와 7-세그먼트에 디스플레이 하고, PC상의 하이퍼 터미널과 ValueCAN-Vehicle Spy 2 아날라이저를 통하여 실시간으로 모니터링 한다.

3.1 CAN 프로토콜 IP 코어의 구성 및 특징

3.1.1 Nios II 개발 보드의 특징

보드의 특징은 다음과 같다[9].

- A CycloneTM EP1C20F400C7 device
- 8 Mbytes of flash memory
- 1 Mbyte of static RAM
- 16 Mbytes of SDRAM
- On-board logic for configuring the Cyclone device from flash memory
- EPCS4 serial configuration device
- On-board Ethernet MAC/PHY device
- Two 5-V-tolerant expansion/prototype headers each with access to 41 Cyclone user I/O pins
- Compact-Flash TM connector header for Type I Compact-Flash(CF) cards
- Mictor connector for hardware and software debug
- Two RS-232 DB9 serial ports
- Four push-button switches connected to Cyclone user I/O pins
- Eight LEDs connected to Stratix user I/O pins

- ▶ Dual 7-segment LED display
- ▶ JTAG connectors to Altera devices via Altera download cables
- ▶ 50 MHz Oscillator and zero-skew clock distribution circuitry

3.1.2 CAN IP 하드웨어 설계

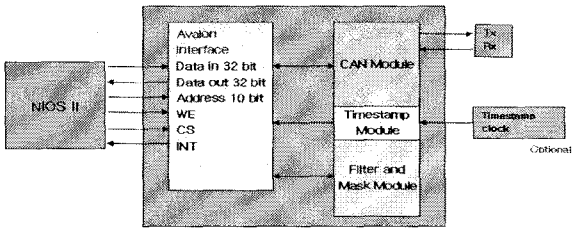


그림 6. Nios II Advanced CAN 구성도
Fig. 6. Structure of Nios II Advanced CAN

본 논문에서 핵심이 되는 CAN 프로토콜 IP 코어는 IFI 사의 IFI_Nios_II_Advanced_CAN 모듈 v6.8을 사용하였다[15][16].

가) IFI_Nios_II_Advanced_CAN IP 코어의 특징[15]~[17]

- ▶ CAN 2.0B
  - Standard or Extended Identifier
  - Remote Frames, Error-Handling
- ▶ Up to 256 message transmit buffer (FIFO Pointer accessible)
- ▶ Up to 256 message receive buffer (FIFO Pointer accessible)
- ▶ Up to 256 message filters(모든 메시지 필터는 one-MASK and one-Identifier-Register를 포함한다.)
- ▶ Nios II embedded processor interface
- ▶ Silent mode
- ▶ High priority messages
- ▶ 32 Bit Time stamp(optional)

나) CAN IP 하드웨어 설계 및 구현

그림 7은 Quartus II의 SOPC Builder GUI를 이용하여 Nios II 프로세서, CAN IP, LED_PIO, BUTTON_PIO, SEVEN_SEG_PIO, UART 등 필요한 하드웨어 인터페이스를 추가하고 생성함을 보인다. Avalon Switch 구조 버스에 다른 외부 디바이스들을 연결하면 자유롭게 자원을 공유할 수 있다[5]~[8].

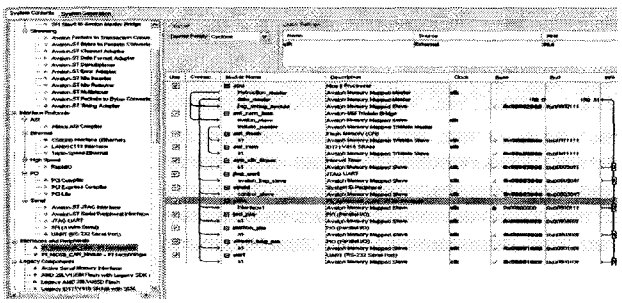


그림 7. SOPC Builder GUI를 이용한 시스템 구성
Fig. 7. System configuration using SOPC Builder GUI

그림 8에서는 SOPC Builder에 추가된 CAN IP 코어의 IP 툴 벤치를 사용하여 내부의 파라미터 값을 설정한다. 타임스탬프 카운터의 사용여부, 사용되는 필터와 마스크의 수, 송수신 FIFO의 크기, CAN 비트 타이밍, CAN 통신 속도, 시스템 클럭을 설정한다.

SOPC Builder에서 사용 할 디바이스들을 연결 하여 시스템을 구성한 뒤, Generate와 컴파일을 하고, 시스템 구성에 사용되는 핀을 할당한다. 핀 할당이 끝난 뒤, 재 컴파일을 하게 되면 시스템 구성에 추가되었던 인터페이스들이 생성된다.

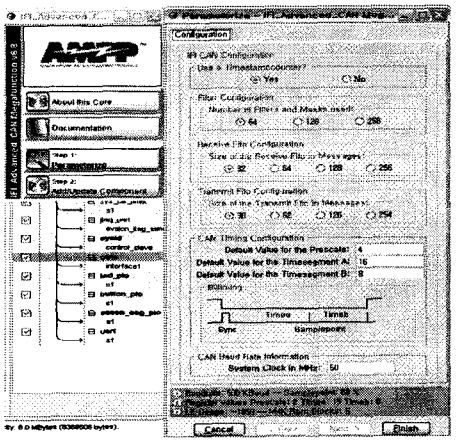


그림 8. IP 툴 벤치를 이용한 CAN IP 코어 내부 파라미터 설정
Fig. 8. Internal parameter settings of CAN IP core using IP tool bench

그림 9에서는 본 논문의 가장 핵심이 되는 CAN 프로토콜 IP 코어가 실장 된 시스템의 전체 설계 회로를 보인다.[1]

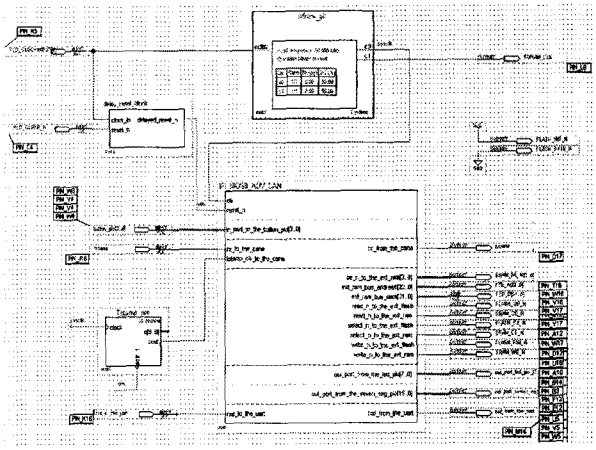


그림 9. IFI_NIOS_II_Advanced_CAN 설계 회로
Fig. 9. IFI_NIOS_II_Advanced_CAN design circuit

3.1.3 응용 소프트웨어 설계 및 구현

CAN IP 하드웨어를 실제로 구동시키는 통합 소프트웨어 환경을 구현한다. 응용 프로그램은 FPGA에 실장 된

CAN IP 코어를 동작시키고, IRQs, 에러 상태, 외부 CAN 보드와 송수신되는 CAN 메시지 데이터를 LED와 7-세그먼트에 디스플레이 한다. 그리고 하이퍼 터미널과 Value CAN-Vehicle Spy 2 아날라이저를 통하여 실시간으로 모니터링 한다. Nios II IDE의 통합 환경 툴을 이용하여 소스코드를 시스템화 하였다[3][4][6].

가) CAN 드라이버 루틴
(ifi_avalon_can_advanced_module.c)

▶ ifi_avalon_can_advanced_module_open()
- CAN 노드의 초기화
- Base, pointer to structure canall_s with timing, interrupt mask, status, mask and filter

▶ ifi_avalon_can_advanced_module_read()
- 버퍼로부터 하나의 메시지를 읽어 들이고, 버퍼 포인터를 증가시킨다.
- Base, pointer to structure canmsg_s with identifier, data[0], dlc
- 성공적으로 읽어 들이면 1을 리턴하고, 에러가 발생하면 -1을 리턴한다.

▶ ifi_avalon_can_advanced_module_write()
- 하나의 메시지를 전송한다.
- Base, pointer to structure canmsg_s with identifier, data[0], dlc
- 성공적으로 읽어 들이면 1을 리턴하고, 에러가 발생하면 -1을 리턴한다.

▶ ifi_avalon_can_advanced_module_wr_int()
- 인터럽트를 활성화 시킨다.
- Base, Interrupt register
- 성공적으로 쓰면 0을 리턴하고, 에러가 발생하면, -1을 리턴한다.

▶ ifi_avalon_can_advanced_module_wr_filter()
- 하나의 단일 mask와 필터쌍으로 쓴다.
- Base, filter number, filter mask, filter identifier
- 성공적으로 쓰면 0을 리턴하고, 에러가 발생하면 -1을 리턴한다.

나) CAN 통신 인터페이스를 위한 응용 프로그램의 구현
(ifi_hello_advanced_can.c)

그림 10은 CAN의 초기화와 비트 타이밍, 인터럽트 설정, 통신 속도, 샘플 포인트, 타이밍 세그먼트 a, 타이밍 세그먼트 b, 프리스케일의 관계를 정의하고 계산하는 부분이다. 이 계산은 하드웨어 설계 시, SOPC Builder상의 CAN IP 코어 툴 벤치에서 설정한 CAN 비트 타이밍 값과 일치해야 한다. 본 논문의 실험에서는 프리스케일 값은 4, 타이밍 세그먼트 a는 16, 타이밍 세그먼트 b는 8, 샘플 포인트는 68%, 통신 속도는 500Kbps, 시스템 메인 클럭은 50MHz로 설정하였다[20]~[22].

CAN IP 코어 노드에 대해 Filter Mask 비트의 값을 '1'로 하면, 이 비트와 수신되는 메시지 ID의 각 비트를 비교하여 자신의 ID와 일치하는 노드만 수신하게 되고, Filter Mask 비트의 값을 '0'으로 하면, 수신되는 메시지 ID의 각 비트 값은 무시되고, 모든 노드의 신호를 수신하게 된다.

```
int ifi_InitCan(struct canall_s *canall)  // HW initialize
  { int baud;
    int timea,timeb,prescale,samplepoint;
    timea    = 16;  // 500kBaud at 50MHz system clock
    timeb    = 8;
    prescale  = 4;

    canall->EPI_CANint =
     IFI_NIOS_CAN_INT_MON_MSK +
     IFI_NIOS_CAN_INT_MRNE_MSK +
     IFI_NIOS_CAN_INT_MEON_MSK;
    canall->EPT_CANtime =
     IFI_NIOS_CAN_TIME_VON_MSK +
     IFI_NIOS_CAN_TIME_NM_MSK +
     IFI_NIOS_CAN_TIME_AON_MSK +
     IFI_NIOS_CAN_TIME_BON_MSK +
    ((prescale-2) <<IFI_NIOS_CAN_TIME_VT_OFST)
    + ((timea-1) <<
    IFI_NIOS_CAN_TIME_WTA_OFST)+((timeb-2));
    prescale = 2 + ((canall->EPT_CANtime &
              IFI_NIOS_CAN_TIME_VT_MSK)
              >> IFI_NIOS_CAN_TIME_VT_OFST);
    timea    = 1 + ((canall->EPT_CANtime &
              IFI_NIOS_CAN_TIME_WTA_MSK)
              >> IFI_NIOS_CAN_TIME_WTA_OFST);
    timeb    = 2 +  (canall->EPT_CANtime &
              IFI_NIOS_CAN_TIME_WTB_MSK);

  baud    = (1 + timea + timeb);
  samplepoint = (100 * (timea + 1)) / baud;
  baud *= prescale;
  baud = (ALT_CPU_FREQ / baud) / 1000;
  return 0;
  }
```

그림 10. CAN의 비트 타이밍과 IRQ 설정 및 초기화
Fig. 10. Bit timing, IRQ setting and initialization of CAN

```
int ifi_InitCanM(struct canall_s *canall)  // ID Filter & Mask
{
/*Filter mask valid all other objects, this Filter MASK will use.*/
/* 1 : 해당 ID만 수신, 0 : 모두 수신 */
    // Filter Mask Valid Bits filtered
    canall -> CANbuffer[0]    = 0xa0000000;
    // ID Valid, Own ID
    canall -> CANbuffer[1]    = 0xa1000000;
    return 0;
```

그림 11. CAN ID Filter & Mask 비트 설정
Fig. 11. CAN ID Filter & Mask bit setting

그림 11에서는 CAN IP 코어 노드의 ID를 0x01로 설정

하였고 이 때, Filter Mask 비트 값을 '0'으로 하여 모든 노드의 신호를 수신하도록 설정하였다.

```
while (1)
 {
  if (capture_a.irqcnt!=capture_a.irqdone)    //interrupt
  {
   printf("CAN Rx ");                  //Receive Info.
   ifi_avalon_can_advanced_read
   (CANA_BASE, &rxmsg_a);  //Incoming message Read
   data = rxmsg_a.CANdata[0]; //Data of Receive message
/*LED_PIO Display*/
IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE,~data);
  ifi_PrCanuart(&rxmsg_a);  //read data display message
   capture_a.irqdone++;
```

그림 12. 수신된 CAN 메시지 처리
Fig. 12. Received CAN message handling

그림 12는 들어오는 인터럽트 수와 이미 처리되어진 인터럽트 수를 비교하여 같지 않을 경우에 수신 인터럽트를 발생하는 부분이다. 인터럽트가 발생되면, 외부로부터 CAN 메시지를 수신하여 해당되는 LED를 제어하고, 현재 수신되는 데이터 값을 7-세그먼트에 디스플레이 한다. 그리고 하이퍼 터미널을 통하여 수신되는 데이터를 실시간으로 모니터링 한다[10][11].

```
/* Tx message CAN ID 0x02 Setting(EXT.) */
obja.EPA_CANid  = 0x22000000;
obja.CANdata[0]  = 0x000000f0; // Tx CANdata[0], Data Setting
obja.EPR_CANdlc = 0x00000001; // Tx CAN Data Length Code
```

그림 13 송신 CAN 메시지 설정
Fig. 13. Transmission CAN message setting

```
intifi_TxCan(alt_u32base,structcanmsg_s*canmsg, intstart)
{
    struct canmsg_s newmsg;
    newmsg.EPA_CANid  = canmsg -> EPA_CANid;
    newmsg.CANdata[0]  = canmsg -> CANdata[0];
    newmsg.EPR_CANdlc = canmsg -> EPR_CANdlc;
    return (ifi_avalon_can_advanced_write(base, &newmsg));
}
```

그림 14 CAN 메시지 송신을 위한 함수
Fig. 14. Function for CAN message transmission

그림 13에서는 CAN IP 코어가 실장 된 Nios II 개발 보드에서 외부 CAN 노드로 송신을 위한 초기의 CAN 메시지 정보(ID, CANdata[0], CANdlc)를 설정한다.

그림 14는 실제 CAN 메시지 송신을 위한 함수이다. 초기에 설정한 송신 메시지 정보로부터 newmsg를 구성하고, 외부 CAN 노드로 이 CAN 메시지를 송신한다.

```
edge_capture =
IORD_ALTERA_AVALON_PIO_DATA(BUTTON_PIO_BASE);
data = edge_capture;
  switch (data)
   {
       case 0x7:
          obja.CANdata[0]=0x00000070;
IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_PIO_BASE,
0x8F81);
          break;
      case 0xb:
          obja.CANdata[0]=0x000000b0;
IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_PIO_BASE,
0xE081);
          break;
       case0xd:
          obja.CANdata[0] = 0x000000d0;
IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_PIO_BASE,
0xC281);
          break;
      case 0xe:
          obja.CANdata[0] = 0x000000e0;
IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_PIO_BASE,
0xB081);
          break;
        default:
          obja.CANdata[0] = 0x000000f0;
IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_PIO_BASE,
 0xB881);
          break;
       }
```

그림 15. 송신 CAN 메시지 디스플레이
Fig. 15. Transmission CAN message display

그림 15는 CAN IP 코어가 실장 된 FPGA 보드에서 버튼 스위치를 누를 때, 해당되는 데이터 값이 전송되어 외부 CAN 노드의 LED를 제어하는 부분이다. 이는 폴링 방식으로 버튼 스위치의 데이터 값을 직접 읽어서 CAN 메시지를 구성하여 전송하고, 그 때 전송되는 데이터 값을 7-세그먼트에 디스플레이 한다[6].

Ⅳ. 실험 및 결과 고찰

본 논문에서의 실험은 Altera사의 Cyclone 디바이스가 장착된 Nios II 개발 보드를 사용하였고, 소프트웨어 툴은 Quartus II 8.0, Nios II IDE 8.0 버전을 사용하였다.

외부의 CAN 노드와 CAN IP 코어가 실장 된 임베디드 프로세서 시스템 간의 통신 인터페이스를 구현하여 송수신 데이터 정보를 LED와 7-세그먼트에 각각 디스플레이 하였다. 그리고 하이퍼 터미널과 Vehicle Spy 2 CAN 아날라이저를 통하여 실시간으로 모니터링 하였다.

4.1 실험 환경

표 2에서는 실험을 위한 하드웨어와 소프트웨어의 환경을 소개한다.

표 2. 실험 환경

Table 2. Experimental environment

▶ 개발 보드 : Altera Cyclone EP1C20F400C7 ▶ CAN IP 코어 버전 : IFI_Nios_II_Advanced_CAN Module(ifi_advanced_can-v6.8) ▶ 외부 CAN 테스트 보드[23] - CAN 컨트롤러 : AT90CAN128(AVR-CAN) ▶ 외부 CAN 트랜시버 보드 - 트랜시버 칩 : Phillips PCA82C250T ※ Nios II 개발 보드에서의 DC +5V 전압 인가, 종단 저항 120 Ohm 연결 ▶ 소프트웨어 툴 : Quartus II 8.0, Nios II IDE 8.0

4.2 CAN IP 코어와 외부 CAN 노드 간 인터페이스 시 고려사항

그림 16의 외부 트랜시버 보드는 CAN IP 코어가 실장된 Nios II 개발 보드에서 송수신되는 CAN 메시지를 물리계층의 CAN 버스상으로 실어주어 외부 CAN 노드와의 인터페이스를 담당하게 된다.

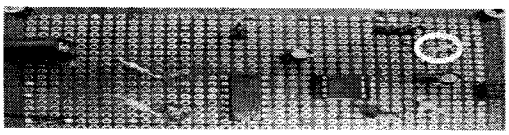


그림 16. 외부 트랜시버

Fig. 16. External transceiver

CAN 버스상에서 정확한 CAN 메시지 송수신을 위해 CAN 버스의 양 끝단에 종단 저항 120 Ohm을 반드시 달아주어야 한다. 그렇지 않으면, 임피던스 반사로인해 송수신 메시지가 왜곡될 수 있다. 그리고 외부 트랜시버의 전원을 CAN IP 코어가 실장 된 Nios II 개발 보드에서 인가해 주어야 한다. 외부에서 전원(DC +5V)을 인가할 경우, 입력 전압 레벨이 다르기 때문에 정확하게 메시지를 송수신 할 수 없다.

2장에서 언급하였듯이, CAN 버스의 각 노드는 자신의 오실레이터로 클럭 되기 때문에 위상이 이동되고, 비트 타이밍 값, 통신 속도 등이 다르다. 따라서 원활한 통신 인터페이스를 위해서는 각 노드 간의 비트 타이밍과 통신 속도를 반드시 일치 시켜주어야 한다. 또한, 오실레이터 허용 오차와 위상 이동을 보상하기 위한 재동기화 과정을 고려하여 네트워크를 구성하여야 한다[19]~[22].

4.3 CAN 통신 인터페이스 구현

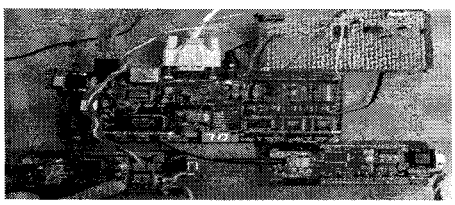


그림 17. CAN 통신 인터페이스 구현(1)

Fig. 17. Implementation of CAN interface(1)

그림 18. CAN 통신 인터페이스 구현(2)

Fig. 18. Implementation of CAN interface(2)

그림 17, 그림 18는 CAN IP 코어가 실장 된 Nios II 개발 보드와 외부 CAN 보드, 그리고 외부 트랜시버를 통해 연결된 CAN 통신을 위한 전체 네트워크의 구성을 보인다.

각 노드의 버튼 스위치 데이터 값을 읽어서 이를 CAN 메시지로 구성하여 송수신한다. 그림 17은 외부 CAN 노드의 CAN 메시지 0x70을 Nios II 개발 보드에서 수신하여 해당되는 값을 7-세그먼트와 LED에 디스플레이 하였고, 그림 18는 CAN IP 코어가 실장 된 Nios II 개발 보드에서의 송신 CAN 메시지 0xd0을 7-세그먼트에 디스플레이하고, 이를 외부 CAN 노드에서 수신하여 해당되는 LED에 디스플레이 하였다

그림 19의 신호를 살펴보면, 외부 CAN 노드와 CAN IP 코어가 실장 된 Nios II 개발 보드에서 송신한 CAN 메시지를 수신 노드에서 에러 없이 정상적으로 수신하고 있음을 알 수 있다. 이와 같이 수신 프레임에 에러가 없으면 수신 노드는 회로적으로 ACK 신호를 발생하고, 송신 노드로 ACK 신호를 재전송하여 통신이 정상적으로 이루어졌음을 확인한다. 이 때, 수신 노드는 자신이 전송하게 될 CAN 메시지 신호를 자체 모니터하기 때문에 전송 할 메시지 신호를 자체적으로 재 수신하여 모니터한다.

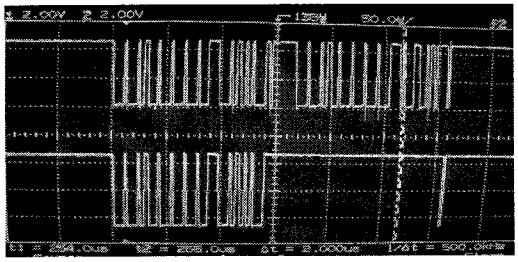


그림 19. CAN 송수신 노드의 신호

(상 : 수신 메시지 / 자체 송신 메시지, 하 : 송신 메시지 / ACK 신호)

Fig. 19. Signals of CAN node

(Above: receiving message / self transmission message, Below : transmission message / ACK signal)

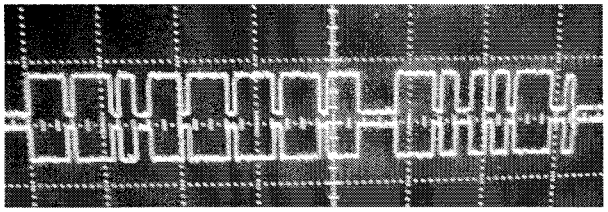


그림 20. CAN 물리계층에서의 CAN_H, CAN_L 신호

Fig. 20. CAN_H, CAN_L signals of CAN physical layer

그림 20은 물리계층의 CAN 버스상으로 전송되는 CAN_High, CAN_Low 두 라인의 CAN 메시지 신호를 보인다. 이와 같이 전압이 다른 두 라인으로 전송을 함으로써 전자기적인 방해요인에 의해 두 라인이 동시에 영향을 받더라도 두 신호의 전압 차는 항상 일정하게 유지되기 때문에 동일한 값의 신호를 전송 할 수가 있다. 그러므로 CAN은 외부 잡음에 강인하고, 더 안정적인 통신이 가능하다.

4.3.1 SignalTap II Logic Analyzer를 통한 CAN 메시지 신호의 분석

그림 21, 그림 22은 Quartus II의 Tools > SignalTap II Logic Analyzer를 이용하여 FPGA 내부 txcan, rxcan 핀에서의 동작 상태를 검증한 것이다.

앞서 3장에서 언급하였듯이, SignalTap II Logic Analyzer는 설계가 실제 조건에서 실행될 때, FPGA 내부의 모든 노드 상태를 JTAG(Joint Test Action Group) 포트를 통하여 접근하는 실제 회로 검증 도구이다.

※ rxcan : 수신 메시지, txcan : ACK

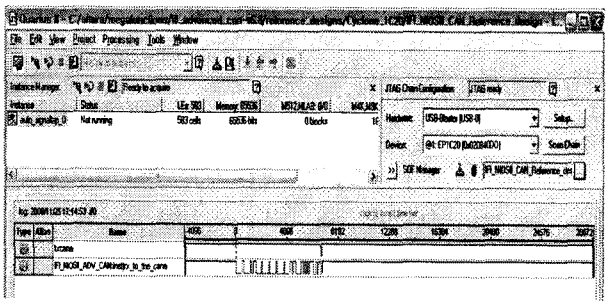


그림 21. SignalTap II Logic Analyzer를 통한 CAN 메시지 신호의 분석(1)

Fig. 21. Analyze of CAN message signal by SignalTap II Logic Analyzer(1)

※ txcan : 송신 메시지, rxcan : 자체 송신 메시지

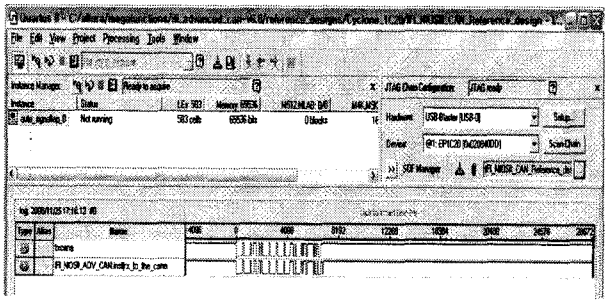


그림 22. SignalTap II Logic Analyzer를 통한 CAN 메시지 신호의 분석(2)

Fig. 22. Analyze of CAN message signal by SignalTap II Logic Analyzer(1)

그림 21은 외부의 CAN 노드에서 송신한 CAN 메시지를 CAN IP 코어가 실장 된 Nios II 개발 보드의 rxcan에서 수신하고, txcan에서 회로적으로 ACK 신호를 발생하여 송신 노드로 재전송함을 보인다. 이로써 정상적으로 수신이 이루어졌음을 확인한다. 그림 22는 Nios II 개발 보드의 txcan에서 송신하는 CAN 메시지 데이터와 송신 할 메시지 신호를 자신의 rxcan에서 재 수신하여 모니터함을 보인다.

4.4 CAN 버스 중재(노드들의 우선순위 결정)

CAN 버스에 두 개 이상의 노드들이 동시에 데이터를 전송할 때, 메시지 식별자의 각 비트들을 충돌 시키는 CSMA/CD 방식으로 우선순위를 결정한다. 이를 위해 각 노드는 11비트 또는 29비트의 고유한 메시지 식별자를 가진다. 본 논문의 실험에서는 메시지 식별자를 29비트의 확장형(CAN 2.0 B)으로 설정하였다.

이와 같은 메커니즘을 기반으로 두 개의 CAN 노드에서 동시에 데이터를 전송 할 때, 높은 우선순위의 노드를 결정하는 방법에 대하여 알아보고자 한다.

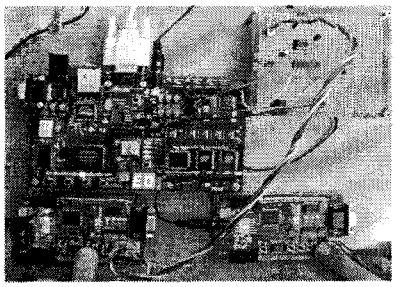
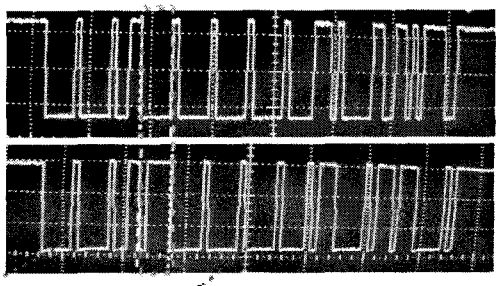


그림 23. 두 개의 CAN 노드에서 동시에 데이터 전송

Fig. 23. Data transmission at the same time from both CAN nodes



	Arbitration Field					Control Field			Data Field				
	32 bits					6 bits			0 to 64 bits				
SOF	식별자 (확장) 11bits	SRR	IDE	식별자 (확장) 18bits	RTR	r1	r2	DLC 4bits	Data Field	CRC 15bits		ACK 2bits	EOF 7bits

그림 24. 각 노드의 송신 데이터 프레임 형태

(상 : ID=0x00, DLC=1, Data=0xe0, 하 : ID=0x0f, DLC=1, Data=0x70)

Fig. 24. Transmission data frame types of each node

(Above: ID:0x00, DLC=1, Data=0xe0, Below: ID:0x0f, DLC=1, Data=0x70)

그림 23은 두 개의 외부 CAN 노드에서 동시에 버튼 스위치를 누를 경우, 우선순위가 높은 노드의 데이터가 먼저 전송되고 있음을 보인다. 각 노드의 송신 메시지 식별자는 좌측 노드는 0x0f, 우측 노드는 0x00으로 설정하였다.

그림 24은 각 노드에서 실제로 송신 하는 CAN 데이터

프레임의 신호를 보인다. 노드들의 우선순위 결정은 중재 필드(Arbitration Field)에서 이루어진다. 이 영역에서 각 비트들을 충돌시켜 가장 높은 우선순위의 노드를 결정하고, 그 때의 데이터를 전송하게 된다[12]~[14][18].

CAN 버스는 'Wired-AND' 로직으로서 '0'의 비트 값이 '1'의 비트 값 보다 항상 우선한다. 위의 데이터 프레임 신호에서 각각의 식별자 비트를 충돌시켜 비교해보면, 0x00의 식별자를 가지는 우측의 노드(그림 24의 송신 데이터 프레임 형태 중, 상)가 높은 우선순위를 가지는 것을 알 수 있다. 그리고 이 노드의 데이터가 먼저 전송을 시작한다.

그림 23에서 실제로 두 노드에서 동시에 데이터를 전송할 때, 0x00의 식별자를 가지는 우측 노드의 데이터(0xe0)가 우선 전송되는 것을 확인하였다. 그리고 우선순위가 높은 노드의 데이터(0xe0)가 먼저 전송이 된 후에 다음 우선순위를 가지는 좌측 노드(0x0f)(그림 24의 송신 데이터 프레임 형태 중, 하)의 데이터(0x70)가 이어서 전송됨을 확인하였다.

4.5 송수신되는 CAN 메시지의 실시간 모니터링

그림 25은 Nios II IDE의 콘솔 창에서 송수신되는 CAN 메시지를 모니터링 한 모습이다. 송수신 노드의 ID, 데이터 길이(DLC), 데이터 정보를 확인 할 수 있다.

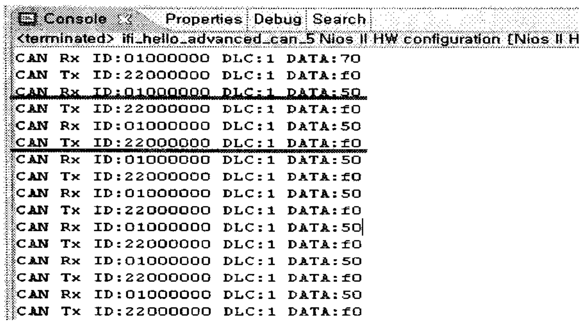


그림 25. Nios II IDE 콘솔 창에서 송수신되는 CAN 메시지

Fig. 25. Transmitted or Received CAN Messages in Nios II IDE consol window

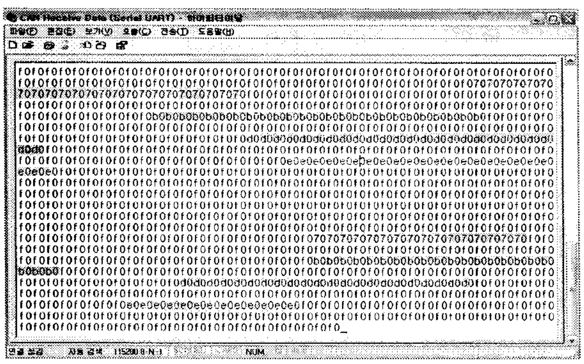


그림 26. 하이퍼 터미널에서 수신된 CAN 데이터

Fig. 26. Receved CAN data through Hyper-Terminal

그림 26는 외부 CAN 노드로부터 CAN IP 코어가 실장된 Nios II 개발 보드에 수신되는 CAN 데이터를 PC상의 하이퍼 터미널에서 실시간으로 모니터링 한 모습을 보인다. 버튼 스위치를 누르지 않았을 때는 0xf0의 데이터가 수신되다가 버튼 스위치의 입력 값에 따라 해당되는 데이터 값(0xe0, 0xd0, 0xb0, 0x70)들이 에러 없이 정상적으로 수신되고 있음을 확인하였다.

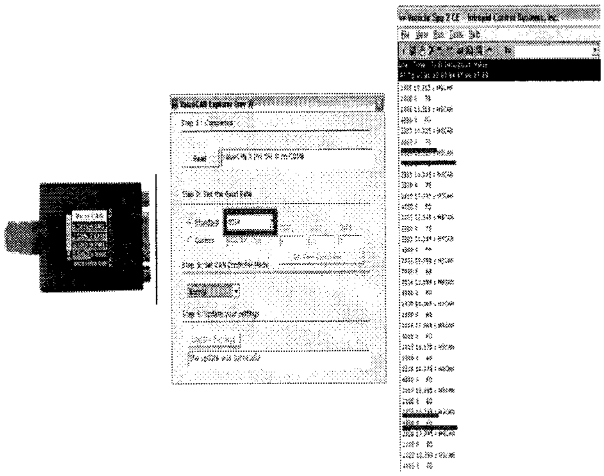


그림 27. ValueCAN-Vehicle Spy 2 아날라이저를 통해 수신된 CAN 데이터

Fig. 27. Receved CAN data through ValueCAN-Vehicle Spy 2 Analyser

그림 27은 ValueCAN과 외부 CAN 노드 간의 CAN 통신 네트워크를 구성하고, Vehicle Spy 2 아날라이저를 통해서 ValueCAN으로 수신되는 CAN 데이터를 실시간으로 수집하는 모습을 보인다.

Vehicle Spy 2 아날라이저는 데이터 수집과 검사, 자동화 기능, 노드 시뮬레이터 기능이 있는 강력한 버스 아날라이저로써 차량 통신 네트워크에서 널리 사용되고 있는 소프트웨어 툴이다[24][25].

V. 결론

기존의 범용 프로세서를 독립적으로 사용하여 구현된 CAN 통신 인터페이스는 다양한 응용의 측면에서 많은 한계점을 드러내었다. 이를 위해 하나의 칩 안에 모든 기능의 구현과 응용이 가능한 FPGA 칩을 사용함으로써 시스템의 소형화와 적응성을 높일 수 있다. 그리고 최근 FPGA 가격이 크게 떨어짐에 따라 경제적인 측면에서도 높은 기대효과를 얻을 수 있다.

본 논문에서 제안한 CAN 프로토콜 IP 코어가 실장 된 FPGA 기반의 임베디드 프로세서 시스템은 Quartus II 툴과 SOPC Builder GUI 환경에서 구성하였다. 그리고 Nios II 개발 보드에서 송수신되는 CAN 메시지의 처리, 실시간 모니터링 방식 등을 Nios II IDE 통합 개발 환경을 이용하여 구현하였다.

Nios II 개발 보드에서 CAN 메시지 송수신을 위해 외부

트랜시버 보드를 제작하고, 외부 CAN 노드와 연결하여 CAN 통신을 위한 네트워크를 구성하였다. 상호 통신 되는 데이터를 LED와 7-세그먼트에 디스플레이 하였고, 하이퍼터미널과 ValueCAN-Vehicle Spy 2 아날라이저를 통해 실시간으로 모니터링 하여 정상적인 통신 과정을 확인하였다. 그리고 서로 다른 메시지 식별자를 가지는 두 개의 CAN 노드에서 동시에 데이터를 전송 할 때, CAN 버스로 전송되는 가장 높은 우선순위의 한 노드를 결정짓는 버스 중재 과정을 확인하였다. 이는 다른 범용 CAN 컨트롤러와도 정확한 동작을 하며 특히, Atmel AT90CAN128과 완벽한 호환성을 보였다.

이에 따라 개발된 CAN 통신 인터페이스는 빠르게 진보하는 하드웨어 기술에 발맞추어 개발자가 보다 쉽게 업그레이드 할 수 있고, IP의 재사용을 가능하게 한다.

향후 연구과제로 좀 더 정확한 통신을 위해 CAN 메시지 송수신 처리 루틴에 대한 프로그램상의 보완과 실시간 모니터링 방식의 차별화에 대한 연구가 보완되어야 할 것이다. 그리고 차량뿐만 아니라 가정이나 산업 자동화, 의료장비, 선박 등 다양한 분야에서 활용이 가능하도록 응용 연구도 함께 진행되어야 할 것이다.

참고문헌

[1] 박영석, "PLD를 이용한 디지털 시스템의 설계," 경남대 지능형홈 인력양성사업팀, pp. 6-13, Feb. 2005

[2] Altera Corp., "Device Family Overview," Available on http://www.altera.com/products/devices, Dec. 2005

[3] Altera Corp, "Nios II Processor Reference Handbook," V8.0, pp. 19-278

[4] Altera Corp, "Quartus II Development Software Handbook," V8.0, pp. 2409-2416

[5] Altera Corp, "Nios II Hardware Development Tutorial," pp. 1-50, May. 2008

[6] Altera Corp, "Nios II Software Developer's Handbook," V8.0, pp. 15-108, May. 2008.

[7] Altera Corp, "AN333 : Developing Peripherals for SOPC Builder," pp. 1-28, May. 2007.

[8] Altera Corp. "Avalon Streaming Interface Specificaion Reference Manual," V1.3, June. 2007.

[9] Altera Corp, "Nios Development Board Reference Manual, Cyclone Edition" pp. 9-54

[10] Altera Corp., "Quartus II Development Software Handbook, V8.1 Volume 5: Embedded Peripherals", UART Core, pp. 71-86, 2008

[11] Altera Corp., "Quartus II Development Software Handbook, V3.0", Nios UART, pp. 1-228, Jan. 2003

[12] "CAN Specification 2.0 part A and B. Robert Bosch Gmbg," pp. 4-9, Sept. 1991.

[13] http://www.can-cia.org

[14] http://www.eskorea.net

[15] http://www.altera.com/products/ip/iup/can/m-ifi-can20b.html?GSA_pos=1&WT.oss_r=1&WT.oss=nios%20CAN

[16] http://www.ifi-pld.de/IP/Advanced_CAN/advanced_can.html

[17] "IFI Nios II Advanced CAN Module User Guide, version rev 6.8," pp. 1-22, Apr. 2008.

[18] Bosch, "CAN specification, version 2.0," pp. 29-30 1991.

[19] Microchip, "A CAN Physical Layer Discussion, AN228," pp. 1-11, 2002

[20] Peter Dzhelekarski, Volker Zerbe, Dimiter Alexiev, "FPGA Implementation of Bit Timing Logic of CAN Controller," IEEE, pp. 214-220, May. 2004.

[21] Bosch, "The Configuration of the CAN Bit Timing," 6th international CAN Conference 2nd to 4th Nov. Turin(Italy) pp. 1-10

[22] Freescale Semiconductor, "CAN Bit Timing Requirements, Stuart Robb, East Kilbride," Scotland. AN1798, pp. 1-15 1999.

[23] Atmel, "8-bit AVR Microcontroller with 32K/64K/128K Bytes of ISP Flash and CAN Controller," AT90CAN128, pp. 11-20, Aug. 2008

[24] http://intrepidcs.com/catalog/product_info.php/cPath/21/products_id/103

[25] http://intrepidcs.com/VehicleSpy/index.html

구 태 묵(Tae-Mook Koo)

2006년 경남대학교 정보통신공학과 (공학사)
2009년 경남대학교 정보통신공학과 (공학석사)
2009년 ~ 현재 코위버(주) 기술연구소 연구원

※관심분야: Embedded Processor System, Optical Transmission Device Design & Development, DWDM(Dense Wavelength Division Multiplexing)

박 영 석(Young-Seak Park)

1979년 영남대학교 전자공학과 (공학사)
1981년 한양대학교 전자공학과(공학석사)
1985년 한양대학교 전자공학과(공학박사)
990년 ~ 1991년 일본 우정성 통신총합연구소(관서선단연구센터) 초빙과학자
1990년 ~ 1991년 일본 긴끼이동통신센터 객원연구원
2001년 ~ 2002년 미국 North Carolina 주립대학(NCSU) 교환교수
1985년 ~ 현재 경남대학교 정보통신공학과 교수

※주관심분야: Software Engineering, Web-based Software Design & Development, Pattern Recognition, Image Processing, Computer Network & Network Computing, Embedded Processor System HW/SW