(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.[7]
H04L 12/00

(45) 공고일자 2005년08월08일
(11) 등록번호 10-0507023
(24) 등록일자 2005년08월01일

(21) 출원번호	10-2000-7001206	(65) 공개번호	10-2001-0022612
(22) 출원일자	2000년02월03일	(43) 공개일자	2001년03월26일
번역문 제출일자	2000년02월03일		
(86) 국제출원번호	PCT/US1998/016369	(87) 국제공개번호	WO 1999/07107
국제출원일자	1998년08월04일	국제공개일자	1999년02월11일

(81) 지정국

국내특허 : 알바니아, 아르메니아, 오스트리아, 오스트레일리아, 아제르바이잔, 보스니아 헤르체고비나, 바르바도스, 불가리아, 브라질, 벨라루스, 캐나다, 스위스, 중국, 쿠바, 체코, 독일, 덴마크, 에스토니아, 스페인, 핀란드, 영국, 그루지야, 헝가리, 이스라엘, 아이슬랜드, 일본, 케냐, 키르키즈스탐, 북한, 대한민국, 카자흐스탄, 세인트루시아, 스리랑카, 리베이라, 레소토, 리투아니아, 룩셈부르크, 라트비아, 몰도바, 마다가스카르, 마케도니아공화국, 몽고, 말라위, 멕시코, 노르웨이, 뉴질랜드, 폴란드, 포르투칼, 루마니아, 러시아, 수단, 스웨덴, 싱가포르, 슬로베니아, 슬로바키아, 시에라리온, 타지키스탄, 투르크맨, 터어키, 트리니아드토바고, 우크라이나, 우간다, 우즈베키스탄, 베트남, 세르비아 앤 몬테네그로, 짐바브웨,

AP ARIPO특허 : 가나, 감비아, 케냐, 레소토, 말라위, 수단, 스와질랜드, 우간다, 짐바브웨,

EA 유라시아특허 : 아르메니아, 아제르바이잔, 벨라루스, 키르키즈스탐, 카자흐스탄, 몰도바, 러시아, 타지키스탄, 투르크맨,

EP 유럽특허 : 오스트리아, 벨기에, 스위스, 사이프러스, 독일, 덴마크, 스페인, 핀란드, 프랑스, 영국, 그리스, 아일랜드, 이탈리아, 룩셈부르크, 모나코, 네덜란드, 포르투칼, 스웨덴,

OA OAPI특허 : 부르키나파소, 베닌, 중앙아프리카, 콩고, 코트디브와르, 카메룬, 가봉, 기니, 말리, 모리타니, 니제르, 세네갈, 차드, 토고,

(30) 우선권주장 08/905,822 1997년08월04일 미국(US)

(73) 특허권자
넥스트 레벨 커뮤니케이션스
미합중국 스테이트 캘리포니아 94928 로너트 파크 스테이트프레임 드라이브 6085

(72) 발명자
구디스티븐에이치.
미국캘리포니아95746그래니트베이프라이어리지5165

(74) 대리인
장용식
정진상
박종혁

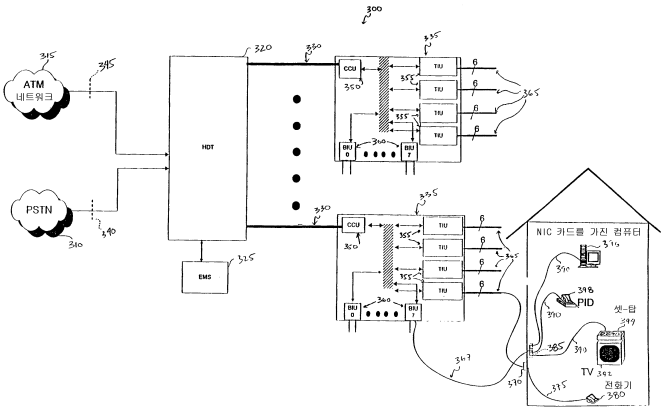
심사관 : 김병우

(54) 신호 스트림으로/으로부터 신호 세트를 삽입/추출하기 위한 방법 및 장치

요약

광 네트워크 유닛에서, 다수의 서로 다른 통신디바이스 세트에 대한 신호를 가지는 신호 스트림에서 하나의 통신디바이스 세트에 대한 신호를 식별하기 위한 방법 및 장치. 각각의 광 네트워크 유닛(ONU)(335)은, 신호 스트림로부터 추출될 필요가 있고 제 1 통신디바이스 세트(392,394,396,398)에 전송될 필요가 있는 신호 세트를 식별하기 위해, 그리고 신호 세트를 제 1 통신디바이스 세트로부터 신호 스트림로 삽입하기 위한 시간주기를 식별하기 위해, 내용주소화 메모리(CAM)를 사용하는 공통제어장치(CCU)(350)를 포함한다.

대표도



색인어

통신디바이스, 신호 스트림, FTTC 네트워크, 비동기식 전송모드(ATM), 텔레포니 신호, 넌-텔레포니 신호, 광 네트워크 유닛(ONU), 멀티플렉스, 디멀티플렉스, 내용주소화 메모리(CAM), SDH식 프레임, 공통제어유닛(CCU)

명세서

배경기술

현재까지, 몇몇의 원격통신 시스템은 동일한 신호 스트림(signal stream)에서 서로 다른 형태의 통신 애플리케이션에 대한 신호를 전송하도록 제안 받아왔다. 몇몇의 이들 시스템은 협대역 애플리케이션에 대한 신호(예를 들면, 텔레포니(telephony) 신호)와 광대역 애플리케이션에 대한 신호(예를 들면, 비디오 및 데이터 신호)를 그들이 전송할 신호 스트림에서 통합한다. 셀에 기초한 포맷(format)에서 통합된 신호 스트림을 전송하는 시스템에서는, 신호 스트림은 비동기식 애플리케이션 신호(즉, 비동기식 통신 애플리케이션을 위한 신호)를 가지는 셀 및 등시성(isochronous) 애플리케이션 신호(즉, 등시성 애플리케이션을 위한 신호)를 가지는 셀의 조합을 포함한다.

서로 다른 형태의 애플리케이션에 대한 서로 다른 형태의 신호를 싣는 통합된 원격통신 시스템은 신호 스트림 내에서 서로 다른 신호를 적절하게 멀티플렉스 및 디멀티플렉스 해야 한다. 예를 들면, 몇몇의 통합된 광대역/협대역 시스템은 비동기 애플리케이션 신호(예를 들면, 비디오 및 데이터 신호)로부터 등시성 애플리케이션 신호(예를 들면, 텔레포니 신호)를 분리해야 한다. 등시성 텔레포니 신호를 추출하기 위한 간단한 선행기술 방법은 메모리에 수신된 프레임에 있는 모든 바이트를 판독하여 그후 등시성 바이트를 판단한다.

도 1 및 도 2는 이러한 선행기술의 한 실시예를 나타낸다. 도 1에 도시된 바와 같이, 선행기술은 수신된 프레임의 모든 바이트를 연속한 바이트-와이드(byte-wide) 기억장소에 순차적으로 저장하는 메모리(105)를 포함한다. 이 선행기술 실시예는 또한 각 기억장소에 대한 포인터를 가지는 포인터의 테이블(110)을 포함한다. 도 2에 도시된 바와 같이, 텔레포니 바이트를 저장하는 기억장소에 대한 각각의 포인터는 특정 전화-인터페이스-유닛(TIU) 카드(205)의 특정 트위스트 페어(twisted pair)(210)를 가리킨다. 따라서, 일단 메모리가 수신된 프레임의 모든 바이트를 완전히 저장하면, 메모리의 등시성 텔레포니 바이트는 포인터의 테이블을 가로질러 적당한 전화라인으로 보내진다.

이 선행기술은 비효율적이며 메모리가 많이 요구된다. 예를 들면, ATM 및 TDM 셀을 가지는 155Mb/s SDH식(SDH-like) 데이터 스트림은 2,340 DS0 바이트를 가진다. 상대적으로 적은 TDM 바이트를 추출하기 위해 이렇게 많은 데이터 바이트를 메모리에 저장하는 것은 매우 비효율적이며 메모리가 많이 요구된다.

그 결과로서, 서로 다른 애플리케이션에 대한 신호를 신호 스트림로부터 효과적으로 디멀티플렉스하는 방법 및 장치가 선행기술에 요구된다. 또한, 서로 다른 애플리케이션에 대한 신호를 신호 스트림로 효과적으로 멀티플렉스하는 방법 및 장치가 요구된다. 더욱 일반적으로는, 다수의 서로 다른 통신디바이스 세트에 대한 신호를 가지는 신호 스트림에서 하나의 통신디바이스 세트에 대한 신호를 식별하는 방법 및 장치가 요구된다.

발명의 개요

본 발명은 다수의 서로 다른 통신디바이스 세트에 대한 신호를 가지는 신호 스트림에서 하나의 통신디바이스 세트에 대한 신호를 식별하는 방법 및 장치를 제공한다. 본 발명의 몇몇 실시예는 다수의 서로 다른 통신디바이스 세트에 대한 신호를 가지는 신호 스트림로부터 제 1 통신디바이스 세트에 대한 신호를 추출하는 방법 및 디바이스이다. 본 발명의 다른 실시예는 제 1 통신디바이스 세트로부터 신호를 신호 스트림로 삽입하는 방법 및 장치이다.

본 발명의 또다른 실시예는 내용주소화 메모리(content addressable memory)를 사용하여 특정한 통신디바이스 세트에 대한 신호 세트를 식별하는, 시분할 멀티플렉싱 및 디멀티플렉싱 방법 및 장치이다. 이에 더하여, 본 발명의 몇몇 실시예는 광섬유 원격통신 네트워크에 사용된다. 이들 실시예는 서로 다른 형태의 애플리케이션에 대한 신호를 가지는 통합된 신호 스트림을 수신하는 광 네트워크 유닛을 포함한다. 광 네트워크 유닛은, 수신된 신호 스트림에서 제 1 형태의 통신애플리케이션에 대한 신호의 위치를 저장하는 내용주소화 메모리를 포함한다.

도면의 간단한 설명

본 발명의 신규한 특징은 첨부되는 청구항에 나타나 있다. 그러나, 예시를 위해서, 본 발명의 몇몇 실시예가 다음 도면을 참고하여 설명되었다.

도 1은, 선행기술인 시분할 디멀티플렉싱 기술에 사용된 메모리 디바이스 및 포인터의 테이블을 나타내는 도면,

도 2는, 도 1의 메모리 디바이스 및 포인터의 테이블을 결합하는데 사용된 4개의 TIU 카드를 나타내는 도면,

도 3은, 본 발명이 구현될 수 있는 파이버-투-더-커브(fiber-to-the-curb) 네트워크를 나타내는 도면,

도 4는, 본 발명의 몇몇 실시예에서 사용된 SDH식 프레임의 한 예를 나타내는 도면,

도 5는, 본 발명의 몇몇 실시예에서 사용된 텔레포니 셀의 한 예를 나타내는 도면,

도 6은, 도 3의 네트워크에 사용되는 공통제어유닛을 나타내는 도면,

도 7은, 본 발명의 한 실시예를 나타내는 도면,

도 8은, 본 발명의 몇몇 실시예에 사용되는 내용주소화 메모리의 한 예를 나타내는 도면,

도 9는, 본 발명의 몇몇 실시예에 사용되는 메모리 인터페이스 유닛의 한 예를 나타내는 도면,

도 10은, 추출된 텔레포니 데이터 및 제어 바이트를 도 3의 네트워크에 있는 4개의 TIU로 전달하는 한 방법을 설명하는 도면,

도 11은, 본 발명의 또다른 실시예를 나타내는 도면.

발명의 상세한 설명

본 발명은, 다수의 서로 다른 통신디바이스 세트에 대한 신호를 가지는 신호 스트림에서 하나의 통신디바이스 세트에 대한 신호를 식별하기 위한 방법 및 장치를 제공한다. 통신 시스템에서, 통신디바이스란 통신 시스템을 통해서 신호를 송신 및/또는 수신하는 디바이스를 말한다. 그러한 통신디바이스의 예들로는 (네트워크 컴퓨터, 퍼스널 컴퓨터, 워크스테이션, 서버 등과 같은) 컴퓨터, 텔레비젼 셋탑 박스(set-top box), 프레미스 인터페이스 디바이스(premises interface device), 및 전화기 등이 있다.

하나 이상의 통신디바이스는 각각의 통신디바이스 세트를 형성한다. 한 세트의 통신디바이스는 유사한 통신디바이스들을 포함할 수도 있다(예를 들면, 한 세트의 전화통신디바이스는 유사한 텔레포니 장치를 사용할 수도 있다). 하나의 통신디바이스 세트란 통신 시스템에 연결하기 위해 유사한 통신 매체를 사용하는 (예를 들면, 트위스트 페어 또는 동축 케이블을 사용하는) 다수의 통신디바이스로서 정의된다. 더하여, 하나의 통신디바이스 세트란 유사한 통신매체 종단 디바이스를 사용하는 (예를 들면, 유사한 라인 카드, 어댑터 카드, 모뎀 등을 사용하는) 다수의 통신디바이스로서 정의될 수 있다.

다음의 설명에서, 많은 세부사항들이 기술된다. 그러나, 당업자는 본 발명이 이러한 세부적 설명없이도 실행될 수 있음을 알 것이다. 다른 경우에는, 필요이상의 상세함으로 본 발명의 설명이 불명확해지지 않도록 주지의 구성이나 장치들이 블록도 형태로서 도시되어 있다.

본 발명의 몇몇 실시예는 파이버-투-더-커브(FTTC) 네트워크에서 실행된다. 그러한 네트워크의 한 예가 도 3에 나타나 있다. 이 도면에 설명되었듯이, FTTC 네트워크(300)는, 하나이상의 네트워크 가입자 사이트(305)에 있는 하나이상의 통신디바이스를 공중교환 원격통신망(PSTN)(310), 비동기식 전송모드(ATM) 네트워크(315), 또는 다른 통신망과 같은 하나 이상의 통신망에 연결한다. 각 네트워크 가입자 사이트(305)는 주거용 또는 상업용 가입자 사이트일 수 있다.

FTTC 네트워크(300)는 가입자 사이트와 PSTN(310), ATM 네트워크(315), 또는 다른 네트워크(예를 들면, 개인적인, 사적인, 또는 비교환공중 네트워크) 사이에서 신호 스트림을 전송한다. 신호 스트림은, 많은 통신디바이스 세트에 송신하는 또는 이로부터 수신받는 신호들을 포함하고 있다. 본 발명의 한 실시예에서는, FTTC 네트워크가 텔레포니 통신디바이스로 및 이들 디바이스로부터 신호를 전송하며 (즉, 텔레포니 애플리케이션 신호), 그리고 넌-텔레포니 통신디바이스로 및 이들 디바이스로부터 신호를 전송한다(즉, 넌-텔레포니 애플리케이션 신호). 본 발명의 또다른 실시예에서는, 다른 종류의 통신디바이스에 대한 신호가 통신 네트워크를 통해서 전송될 수 있다.

FTTC 네트워크(300)는 호스트 디지털 터미널(HDT)(320), 엘리먼트 관리시스템(EMS)(325), 다수의 광섬유 케이블(330), 및 다수의 광 네트워크 유닛(ONU)(335)을 포함한다. 도 3에 나타내었듯이, HDT는 PSTN, ATM 네트워크, 및/또는 다른 네트워크에 연결되어 있다. 본 발명의 한 실시예에서, PSTN-HDT 인터페이스(340)는 벨코어 사양(Bellcore specification) TR-TSY-000008, TR-NWT-000057, 또는 TR-NWT-OOO303과 같은, 표준 본체 중의 하나에 의해 채택된 사양을 따른다. PSTN에서 물리적 인터페이스는 DS1 신호를 전달하는 트위스트 페어 또는 OC-3 광신호를 전달하는 광섬유 케이블이 될 수 있다.

ATM 네트워크-HDT 인터페이스(345)는 (OC-3, OC-12c 등과 같은) 광학적 인터페이스를 사용하여 실현될 수 있다. 본 발명의 한 실시예에서, HDT(320)는 ATM 셀을 전달하는 신호를 수신만 하는 두개의 광 브로드캐스트 포트 및 신호를 송수신하는 한개의 광학적 쌍방향 포트를 가진다.

HDT는 PSTN, ATM 네트워크, 또는 다른 네트워크로부터 다운스트림(downstream) 신호를 수신받고, 이들 신호를 다운스트림의 네트워크 가입자 사이트로 라우트(route)한다. 또한 네트워크 가입자 사이트로부터 업스트림(upstream) 신호를 수신받고, 이들 업스트림 신호를 PSTN, ATM 네트워크, 또는 다른 네트워크로 라우트한다.

네트워크의 다운스트림 측면에서, HDT는 멀티플렉서로서 역할을 하는데, (1) 수신된 다운스트림신호(즉, PSTN, ATM 네트워크, 또는 다른 네트워크로부터 수신된 신호)의 전송매체로의 액세스를 제어하며, (2) 수신된 신호를 적당한 광섬유 케이블(330)을 통하여 적당한 ONU(335)로 전송한다. 반대로, 네트워크의 업스트림 측면에서, HDT는 디멀티플렉서로서의 역할도 하는데, (1) 수신된 업스트림신호(즉, ONU로부터 수신된 신호)를 PSTN, ATM 네트워크, 또는 다른 네트워크에 대한 신호 스트림로 파스(parse)하며, (2) 이 신호 스트림을 적당한 네트워크로 보낸다.

HDT는 또한 다운스트림 및 업스트림 신호를 이들의 목적지로 보내기 전에 재포맷(re-format)한다. 아래에서 더 설명하듯이, HDT는 다음과 같은 포맷팅 동작을 실행한다 : 등시성 및 비동기식 신호를 프레임에 결합하며, 통합된 프레임을 개별

적인 신호 스트림으로 파스하며, 신호속도를 조정하며, 및 전기적 신호와 광신호를 상호변환한다. 당업자는, 본 발명의 다른 실시예에서는 HDT가 이러한 모든 포맷팅 동작을 다 실행하는 것은 아닐수도 있으며, 또는 더 부가적인 동작까지도 실행할 수 있다는 것을 알 것이다.

도 3에 도시된 FTTC 네트워크에서, HDT는 (데이터 및 비디오 신호와 같은) 비동기식 애플리케이션 신호를 가지는 셀과 (텔레포니 신호와 같은) 등시성 애플리케이션 신호를 가지는 셀의 조합을 포함하는 프레임에, 수신된 다운스트림신호를 결합한다(즉 맵(map)한다). HDT는 이들 프레임을 미리 특정된 속도로(예를 들면, 125㎲마다 한번씩) ONU로 전송한다. 또한, HDT는 수신된 업스트림 프레임을 PSTN, ATM, 또는 다른 네트워크에 공급되는 개별적인 바이트들로 파스한다.

도 4는 광섬유 케이블(330)을 가로질러 전송되는 프레임의 한 예를 나타낸다. 도시된 각 프레임은 오버헤드 및 페이로드 엔벨로프(envelope)를 포함하는 SDH식 프레임이다. 페이로드 엔벨로프는 41개의 비동기식 및 등시성 정보 셀을 가진다. 각 셀은 57바이트를 가진다.

도 5는 등시성 텔레포니 셀의 한 예를 나타낸다. 도시된 바와 같이, 각 셀은 세개의 예약(reserved) 바이트(R), 및 여섯개의 DS0 바이트 그룹을 포함한다. 도 5는 각각의 그룹을, 하나의 시그널링 바이트(S)를 뒤따르는 8개의 DS0 데이터 바이트로 나타낸다. 시그널링 바이트는 8개의 DS0 바이트에 대한 상태와 제어신호(예를 들면, 온/오프 후크, 링(ring) 등)를 포함한다. 선택적으로, 각 그룹은 정보 및 제어신호의 혼합체를 가지는 9개의 클리어(clear) DS0 바이트를 포함할 수도 있다.

또한, HDT는 수신받는 데이터의 신호속도를 조정한다. 예를 들면, 본 발명의 한 실시예에서, HDT(320)는 PSTN(310)으로부터 DS1 속도로 수신하는 신호를 DS0 속도의 신호로 변환한다; 그후 이들 신호를 SDH식 프레임으로 ONU로 전송한다. 업스트림측에 대해서, HDT는 역신호속도 변형연산(inverse signal rate transformation operation)을 수행한다. 또한, HDT는 디지털 신호의 전기적 표현을 광학적 표현으로 변환하며, 그 반대로도 행한다.

도 3에 나타내었듯이, HDT는 EMS(325)에 연결된다. EMS는 HDT(320)가 위치한 중앙사무소에서, 옥외에서, 또는 거주지에서 FTTC 네트워크의 서비스 및 장치를 제공하는데 사용된다(예를 들면, TIU와 분리된 트위스트 페어를 할당한다). EMS는 소프트웨어에 기초하며 퍼스널 컴퓨터 상에서 작동할 수도 있으며, 이 경우에 하나의 HDT 및 이에 연결된 결합 액세스 네트워크 장치(associated access network equipment)를 지원한다. 선택적으로, EMS는 워크스테이션 상에서 작동할 수도 있는데, 이 경우에는 다중 HDT 및 액세스 네트워크를 지원한다.

광섬유 케이블(330)은 서비스받는 다수의 지역에 위치한 다수의 ONU(335)에 HDT를 통신상으로 연결한다. 본 발명의 몇몇 실시예는 ONU와 HDT 사이의 통신을 위해 양방향 단일모드 광섬유 라인 및 2중 파장 전송방식을 사용한다. 더우기, 본 발명의 몇몇 실시예는 64개의 ONU에 HDT를 연결하기 위해 64개의 광섬유 케이블을 사용한다.

도 3에 나타내었듯이, 각 ONU는 공통제어유닛(CCU)(350), 4개의 전화 인터페이스 유닛(TIU)(355), 및 8개의 광대역 인터페이스 유닛(BIU)(360)을 포함한다. CCU는 ONU의 동작을 제어한다. 특히, CCU는 HDT에 의해 실행된 것과 유사한 포맷팅 동작을 실행한다.

예를 들면, CCU는 (1) 수신된 다운스트림 신호(즉, HDT로부터 수신한 프레임)를 텔레포니 애플리케이션 신호 스트림 및 넌-텔레포니 애플리케이션 신호 스트림로 파스하며, 및 (2) 텔레포니 애플리케이션 신호 스트림을 적당한 TIU(355)에 공급한다. 또한 CCU는 TIU(355) 및 BIU(360)로부터 수신한 신호 스트림을 SDH식 프레임(도 4와 관련하여 위에서 설명한 것과 같은 프레임)에 결합하며, 그 후 광섬유 케이블(330)을 통해 HDT로 전송한다. CCU가 위에서 설명된 파싱(즉, 디멀티플렉싱) 및 결합(즉, 멀티플렉싱) 동작을 수행하도록, 본 발명의 다른 실시예가 CCU에서 사용될 수 있다. 두개의 그러한 실시예가 도 7 및 도 11과 관련하여 아래에서 설명될 것이다.

도 6은 CCU(350)의 한 예를 나타낸다. 도시된 것과 같이, CCU는 양방향(BIDI) 광 변환기(605), 프레임기(610), TIU 인터페이스(TIUI)(615), BIU 인터페이스(BIUI)(620), 및 마이크로프로세서(625)를 포함한다. 다운스트림 측면에서, 광 변환기(605)는 광섬유 케이블(330)을 통해 수신한 광 신호를 전기적 직렬 신호 스트림로 변환한다. 그후 프레임기(610)는 이 비트-와이드 전기적 신호 스트림을 수신하여 이를 바이트-와이드 신호 스트림로 변환한다. 업스트림 측면에서, 광 변환기는 비트-와이드 전기적 신호를 프레임기로부터 수신하며, 광섬유 케이블(330)을 통한 전송을 위해 이들 전기적 신호를 광 신호로 변환한다.

또한 프레임기는 페이로드 및 오버헤드 바이트의 위치를 결정하는데 사용되는 프레임 경계 정보를 판단하기 위해, 들어오는 신호 스트림을 스캔한다. 일단 프레임기가 프레임 경계를 결정하면, 각 프레임의 시작을 알리는 프레임 동기(synch) 신호를 발생시킨다. 아래에서 설명하는 바와 같이, 이 프레임 동기 신호는 다수의 CCU의 회로에 의해 사용된다.

프레임기는 TIUI로부터 신호를 송신 및 수신하기 위해서 통신상으로 TIUI(615)에 결합되어 있다. TIUI가 다운스트림 신호 스트림로부터의 텔레포니 애플리케이션 신호를 파스하고 텔레포니 애플리케이션 신호를 업스트림 신호 스트림에 결합하도록 하기위해, 본 발명의 몇몇 실시예는 TIUI에서 실행된다. 두개의 그러한 실시예가 아래의 도 7 및 도 11과 관련하여 설명될 것이다.

프레임기는 또한 BIUI를 통해서도 신호를 송수신한다. BIUI는 관리기능(예를 들면, 패러티 체크 및 오버헤드 부가 기능)을 수행한다. 다운스트림 측면에서, BIUI는 프레임기로부터 수신한 신호를 BIUs(360)로 전달한다. 그후 각각의 개별 BIU는 이곳으로 어드레스 되어있는 수신신호의 부분들을 추출하며, BIU는 추출된 신호를 네트워크 가입자 사이트(305)로 보낸다. 업스트림 측면에서, BIUI(620)는 BIU에 대해서 중재자로서의 역할을 한다. 이 능력으로, BIUI는 각 프레임 페이로드의 다른 부분들을 다른 BIU에 배치한다.

CCU(350)의 마이크로프로세서(625)는 CCU의 다양한 요소를 프로그램하는데 사용된다. 아래에서 더 설명되는 바와 같이, 예를 들면, 마이크로프로세서는 TIUI의 다양한 요소를 프로그램하는데 사용된다. 이 프로그래밍을 통해서, TIUI는 수신또는 송신된 신호 스트림에 텔레포니 바이트의 위치를 저장할 수 있다.

도 3에서, 각각의 ONU(335)는 백플레인(backplane) 상호접속을 통해 TIUI(615)에 결합된 4개의 TIU(355)를 포함한다. 각 TIU(355)는 6개의 트위스트 페어(365)에 연결되어 있고, 이로서 각 ONU는 24개의 네트워크 가입자 사이트에 서비스를 제공한다. 본 발명의 다른 실시예에서, FTTC 네트워크(300)는 다른 개수의 TIU 및/또는 각 TIU에 대한 다른 개수의 트위스트 페어를 가지는 ONU를 이용한다.

TIU가 CCU로부터 신호 스트림을 수신하면, TIU는 신호 스트림을 (기존전화 서비스 신호(POTS), 코인(coin), ISDN, 등과 같은) 협대역 서비스를 위한 신호 스트림로 변환한다. 예를 들면, TIU는 아날로그 POTS를 발생시켜서 트위스트 페어(365)를 통해 네트워크 가입자 사이트(305)로 전송한다. 네트워크 가입자 사이트(305)에서는, 네트워크 인터페이스 디바이스(NID)(370)가 트위스트 페어(365) 및 전화기(380)에 연결된 가입자 사이트 트위스트 페어(375) 사이의 인터페이스 및 경계점으로서의 역할을 한다. NID는 고전압 보호기능을 제공한다. 더우기, NID(370) 및 트위스트 페어(365 및 375)를 통해, 각각의 TIU(355)는 네트워크 가입자 사이트로부터 아날로그 신호를 수신한다. 그후 이들 아날로그 신호를 CCU로 공급하기 위해 디지털 신호로 변환한다.

또한 각 ONU(335)는 백플레인 상호접속을 통해 CCU(350)에 연결된 8개의 BIU(360)를 가진다. 각각의 BIU(360)는 두개의 동축케이블에 연결된다. 다른 개수의 BIU 및/또는 동축케이블이 또한 ONU에 사용될 수도 있다. 위에서 설명한바와 같이, 각 BIU는 CCU(350)의 BIUI(620)에 통신상으로 결합되어 있다. 이 결합을 통해서, BIU는 CCU로부터 신호를 수신한다.

BIU는, 수신된 셀이 BIU로 어드레스 되었는지를 결정하기 위해 각 수신된 셀의 오버헤드 바이트의 일부분을 디코드하는 필터를 포함한다. 만일 그렇다면, 필터는 또한 BIU의 어드레스된 동축케이블을 결정한다. 그후 필터는 수신된 셀을 재포맷하며 FIFO 큐(queue)에 저장한다. FIFO의 내용들이 판독되면, BIU는 판독된 신호를 RF 반송파 상에서 변조하며, 어드레스된 동축케이블을 통해 네트워크 가입자 사이트(305)에 있는 분배기(splitter)(385)로 데이터를 전송한다.

가입자 사이트 동축케이블(390)은 분배기를 네트워크 가입자 사이트의 다수의 통신 장치에 연결한다. 통신 장치는 텔레비젼 셋-탑(394)을 가진 텔레비젼(TV)(392), 네트워크 인터페이스 카드(NIC)(396)를 가진 컴퓨터, 및 전화기에 연결된 프레미스 인터페이스 디바이스(PID)(398)를 포함한다. 이들 디바이스의 각각은 비디오, 데이터, 및 음성 신호를 얻기 위해, 수신된 재포맷된 셀을 파스 및 디코드한다.

특히, 각 통신 장치는 인터페이스 서브 시스템을 필요로 하는데, 이것은 가입자 사이트 동축케이블에 있는 포맷으로부터 장치가 요구하는 서비스 인터페이스로 신호의 변환을 제공한다. PID(398)는 가입자 사이트 동축케이블(390)에 실려온 음성 신호를 추출하여 전화기와 호환가능한 전화용 신호를 발생시킨다. 유사하게, 텔레비젼 셋-탑(394)은 디지털 비디오 신호를 TV와 호환가능한 아날로그 신호로 변환한다. NIC 카드는 컴퓨터 호환 신호를 발생시킨다.

또한, 가입자 사이트 동축케이블(390), 분배기(385), 및 동축케이블(367)을 통하여, BIU는 FTTC 네트워크(300)를 통한 업스트림 전송을 위한 광대역 신호를 수신한다. BIU는 수신한 신호를 복조하며, 광대역 신호의 결과 성분을 단일 광대역 신호 스트림로 결합하여 BIUI(620)에 중계한다.

당업자는 본 발명이 다른 FTTC 네트워크에서도 구현될 수 있음을 알 것이다. 예를 들면, 본 발명의 몇몇 실시예는 미국에서 1997년 8월 4일 출원되어 현재 출원계속 중인 출원번호 제___호, 발명자는 토마스 R. 에머스(Thomas R.Eames), 속달우편 라벨 번호 EH385539499US를 가지는, 발명의 명칭이 "단일 액세스 플랫폼(Unified Access Platform)"인 출원에 공개된 네트워크에서도 구현될 수 있다. 이 애플리케이션의 설명부분이 본 애플리케이션에 참고적으로 포함되어 있다. 선택적으로, 당업자는 다른 네트워크에서도 본 발명이 구현될 수 있음을 알 수 있다.

FTTC 네트워크(300)에서 실시되는 본 발명의 하나의 실시예는, 한 세트의 텔레포니 통신디바이스에 대한 신호를, FTTC 네트워크를 통해 전송된 통합된 텔레포니/넌-텔레포니 신호 스트림로부터 추출한다. 이 실시예는 위에서 언급한 바와 같이 SDH식 프레임에 구성되는 수신된 신호 스트림에서 텔레포니 애플리케이션 신호의 위치를 식별하는 위치식별 지시자(indicia)(예를 들면, 포인터)를 저장한다.

그후 전송된 프레임을 수신할 때 위치식별 지시자를 발생하며, 동시에 이 지시자를 저장된 하나이상의 지시자와 비교한다. 발생된 지시자가 저장된 지시자에 매치하면, 프로세스는 발생된 지시자에 대응하는 텔레포니 신호를 수신 신호 스트림로부터 추출한다. 그후 이 프로세스는, 추출된 신호 세트를 통신디바이스 세트로 보내기 위해, 매치된 저장 위치에 기초해 적당한 라우팅(routing) 신호를 발생시킨다.

도 7은 그러한 한 예를 나타낸다. 이 장치는 CCU의 TIUI(615)에서 구현되며, ONU(335)의 TIUs에 대한 인터페이스로서의 역할을 한다. 이 인터페이스는 SDH식 19.44MHz의 속도에서 바이트-와이드 증분으로 SDH식 프레임을 계속해서 수신한다. 인터페이스는 바이트 카운터(705), 내용주소화 메모리(710), 메모리 인터페이스 유닛(715), 스윙(swing) 데이터 랜덤 액세스 메모리(720), 제어 랜덤 액세스 메모리(RAM)(725), 다운스트림 TIU 인터페이스(730), 포인터의 테이블(735), 포인터 램(RAM) 제어기(740), 비트 카운터(도시생략), 마이크로 프로세서 인터페이스 유닛(745), 및 다운스트림 동기화 유닛(도시생략)을 포함한다.

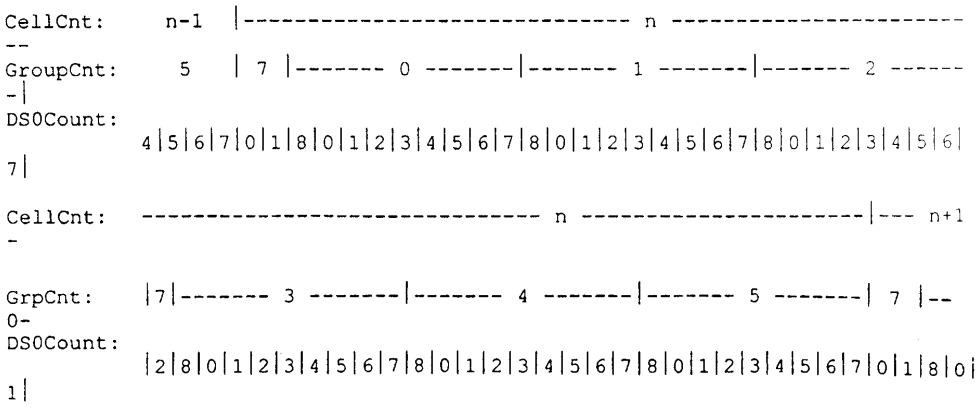
카운터(705) 및 내용주소화 메모리(CAM)(710)는 공동으로, 메모리 인터페이스 유닛(715)에 보내지는 다운스트림 SDH식 프레임에서 텔레포니 애플리케이션 신호를 식별하기 위해 동작한다. 특히, 카운터는 각 프레임의 정렬을 유지하기 위해서 (프레임기(610)에 의해 발생되는) 프레임 동기화 신호를 수신한다. 또한 카운터는 다운스트림 동기장치(도시생략)로부터 동기화 신호를 수신한다. 이 동기화 신호는 SDH-영역에서 카운터 및 텔레포니-영역에서 카운터를 동기로 유지하며, 그 결과 이들 영역에서 인터페이스 유닛(715 및 730)이 메모리(720 및 725)를 동시에 액세스하지 못한다.

또한 카운터는 19.44MHz SDH-바이트 클록을 수신하며, 이 주파수에서 바이트-와이드 세트에 수신되는 신호와 동기인 3개의 카운트 값을 발생시킨다. 즉, 카운터는 카운트 값의 세트를 발생시키는 바이트 카운터이다. 카운트 값의 각 세트는 포인터로서의 역할을 하여, 메모리 인터페이스 유닛(715)에 수신된 프레임 내에 바이트의 위치를 지정한다. 각 포인터는, 포인터에 대응하는 수신 바이트가 텔레포니 애플리케이션 바이트인지를 판단하기 위해 CAM에 입력된다.

도 4 및 도 5와 관련하여 위에서 설명하였듯이, 본 발명의 몇몇 실시예는, 텔레포니 애플리케이션 신호(예를 들면, TDM 신호) 및 넌-텔레포니 애플리케이션 신호(예를 들면, ATM 신호)의 41개 셀의 페이로드를 가지는 SDH식 프레임을 멀티플렉스 또는 디멀티플렉스하도록 설계된다. 각 텔레포니 셀은 9바이트씩 6그룹으로 구성된다. 9바이트는 하나의 시그널링 바이트 뒤에 8 DS0 바이트가 따르도록 배열된다.

이러한 셀 구조가 주어지면, 카운터는 (1) 현재 수신되는 바이트의 셀 카운트를 지정하는 셀 카운트 값(RxCellCnt), (2) 현재 수신되는 바이트의 그룹 카운트를 지정하는 그룹 카운트 값(RxGrpCnt), 및 (3) 현재 수신되는 바이트의 바이트 카운트를 지정하는 DS0 바이트 카운트(RxDS0Cnt) 를 발생시킨다. 이런 방법에서, 3개의 카운트 값은 41개의 포텐셜 TDM 셀의 각각에 있는 모든 바이트를 유일하게 식별하는 포인터로서의 역할을 한다.

이들 3개 카운트의 동작이 아래에 도시되어 있다.



각 그룹내에 있는 DS0 숫자 8은 수퍼-프레임, 멀티플렉스된(super-frame, multiplexed) 시그널링 바이트로 사용된다. 또한, 각 셀 내에는 0, 1, 또는 2의 DS0 카운트를 가지는 그룹 7에 대응하는 3개의 예약 바이트가 있다.

도 7에 있듯이, CAM(710)은 발생된 카운트 값(RxCellCnt, RxGrpCnt, RxDS0Cnt)을 수신하며, 수신된 셀의 타입을 표시하는 신호(RxCellType)를 수신한다. CAM은, 제시된 어떤 신호 세트(예를 들면, 데이터 워드)를 모든 CAM 내용과 한번에 비교하는 능력 및 제시된 신호 세트와 내용이 매치하는 CAM에서의 위치를 보고하는 능력을 가진 메모리 디바이스이다. 선행기술에서 CAM은, 수신된 패킷이 브릿지 또는 라우터(router)에 연결된 LAN에 어드레스되었는지 판단할 목적으로, 추출된 MAC 어드레스와 저장된 MAC 어드레스를 비교하기 위해 브릿지 및 라우터에 사용되었다.

도 8은 본 발명의 몇몇 실시예에 사용되는 CAM(710)의 한 예를 나타낸다. 도시되어 있듯이, CAM은 제어회로(805), 제 1 메모리(810), 제 2 메모리(815), 다운스트림 비교기(820), 및 디코더(825)를 포함한다. 비록이 도면은 두개의 메모리를 표현하지만, 이 표현은 설명의 용이함을 위해 CAM을 단순하게 그림으로 나타낸 것이며 두개의 메모리는 단일 메모리 어레이의 부분이 될 수도 있음을 당업자는 인식할 것이다.

마이크로프로세서 인터페이스 유닛(745) 및 제어회로(805)를 통해, 마이크로프로세서(625)에 연결된 제 1 및 제 2 메모리는 저장을 위해 데이터 신호(즉, 위치식별 지시자) 및 제어 신호(예를 들면, 블록 크기 지시자)를 각각 수신한다. 특히, 이 연결을 통해, 제 1 메모리(810)는 각 SDH식 프레임에 있는 32 텔레포니 애플리케이션 바이트의 위치를 식별하는 32개의 위치식별 지시자를 수신 및 저장한다.

본 발명의 어떤 실시예에서, 제 1 메모리는 30개의 위치식별 지시자를 저장하며, 이때의 각 위치식별 지시자는 메모리 인터페이스 유닛에 수신된 각 프레임 내에 있는 텔레포니 애플리케이션 바이트의 위치를 지정하는 포인터이다. 프레임이 도 4에 도시된 구조를 가질 경우, 그리고 텔레포니 셀이 도 5에 도시된 구조를 가질 경우, 각 저장된 포인터는 13비트를 가질 수 있다. 이들 비트 중에 6비트는 셀 값을 식별하며, 3비트는 그룹 값을 식별하며, 그리고 4비트는 바이트 값을 식별한다.

제 2 메모리(815)에서, 마이크로프로세서는 신호 스트림에서 추출한 신호 블록의 크기를 결정하는 제어비트를 저장한다. 본 발명의 몇몇 실시예에서, 제 2 메모리는 32개의 제어비트 세트를 저장하며, 그 결과 제 1 메모리의 각 포인터에 대해 하나의 제어비트 세트가 있다. 각각의 제어비트 세트는 2비트를 포함한다. 수신된 바이트가 수신된 프레임에서의 그 위치를 제 1 메모리에 저장된 포인터 중의 하나에 매치시킬 때, 메모리 인터페이스 유닛은 매치된 포인터에 대응하는 두개의 제어비트를 디코드하는데, 수신된 바이트를 추출하는것에 더하여 수신된 신호 스트림에서 수신된 바이트를 뒤따르는 1 내지 3 바이트를 추출할 필요가 있는지를 결정하기 위해서이다.

위에서 설명한바와 같이, CAM은 또한 32개의 비교기(820)를 포함한다. 각 비교기는 제 1 메모리에 저장된 개개의 13비트 포인터를 바이트 카운터(705)의 13비트 출력과 비교한다. 예를 들면, 제 1 메모리 어레이의 n번째 행에 대한 비교기는 n행(Rn)과 0내지 12의 열(C0 및 C12)에 저장된 비트를 수신하며, 이들 비트를 발생된 13 카운트 비트(RxCellCnt, RxGrpCnt, 및 RxDS0Cnt)와 비교한다.

각 비교기는 13개의 X-NOR 게이트를 포함하는데, 각 X-NOR 게이트는 저장된 포인터 중의 1비트 및 발생된 카운트 값 중의 1비트를 수신한다. 각 비교기는 또한 비교기의 13개 X-NOR 게이트의 출력을 수신하는 AND 게이트(도시생략)를 가

진다. 각 AND 게이트의 출력은 비교기의 출력을 나타낸다. 따라서, 32개의 비교기(820)는 32개의 출력라인을 가진다. 수신된 바이트의 위치가 저장된 포인터와 매치할 때, 포인터의 비교기는 그 출력라인을 하이(high)로 함으로써 히트(hit)(즉, 매치)를 표시한다. 당업자는 선택적인 CAM 구조가 본 발명의 다른 실시예에서 사용될 수 있음을 알 것이다.

아래에서 더 상술하는 바와 같이, 메모리 인터페이스 유닛(715)은 비교기(820)의 32비트 출력을 수신한다. 이 인터페이스 유닛은 우선순위 인코더(priority encoder)를 포함하는데, 이것은 제 1 메모리의 매치된 포인터를 저장한 행의 어드레스를 나타내는 5비트 출력을 발생시키기 위해서 32비트 출력을 수신한다. 매치된 포인터에 대한 행 어드레스는 제 2 메모리의 제어 신호의 대응하는 쌍에 대한 행 어드레스와 동일하다. 따라서, 5비트 어드레스는 제 2 메모리의 디코더(825)에 공급된다. 그후 이 디코더는 수신된 어드레스 신호에 의해 식별된 행에 저장된 2개의 제어비트를 래치 및 출력한다. 그후 메모리 인터페이스 유닛은 이 2비트를 수신하며, 아래에서 설명하듯이, 이것은 인터페이스 유닛이 수신된 바이트를 뒤따르는 3바이트를 추출하도록 지시한다.

또한, CAM은 (RxCellType 또는 TxCellType와 같은) CellType 신호를 수신한다. 이 신호는 수신된 프레임에서 현재 수신되고 있는 셀이 텔레포니 셀인지 또는 넌-텔레포니 셀인지를 표시한다. 이 신호를 발생시키기 위해, CCU는 프레임 페이로드에 있는 41개의 셀에 대한 41개의 레지스터를 가진 메모리(도시생략)를 포함한다. 각 레지스터는 대응하는 셀이 텔레포니 셀인지를 표시하는 CellType 플래그(flag)를 저장한다. 카운터(705)에서 발생된 셀 카운트 어드레스를 사용하여, CCU는 연속적으로 41개의 레지스터를 판독하여 CellType 신호를 발생시킨다. 셀이 텔레포니 셀이 아닐때, CellType 신호의 값은 비교기(820)를 파워 다운(power down)한다.

상술한 바와같이, 메모리 인터페이스 유닛(715)은 다운스트림 신호 스트림을 수신한다. 또한 CAM과 동기적으로 동작하기 위해서, 발생된 카운트 값(RxGrpCnt 및 RxDS0Cnt), 및 RxCellType을 수신한다. 이 유닛은 또한 CAM(710)과 연결되어 32비트 출력을 수신한다. 이 출력에 기초하여, CAM에서 매치된 포인터의 5비트 어드레스를 발생시킨다. 그후 인터페이스 유닛은 CAM으로부터 매치된 포인터에 대응하는 2 제어비트를 검색하기 위해 이 어드레스를 사용한다. CAM의 32 출력비트 뿐만 아니라 2 제어비트에 기초하여, 인터페이스 유닛은 1 내지 4의 텔레포니 바이트를 수신된 신호 스트림로부터 추출하며, 및 어드레스 및 제어 신호를 발생시켜서, 추출된 텔레포니 바이트를 데이터 및 제어 RAM(725 및 730)에 저장한다.

도 9는 메모리 인터페이스 유닛(715)의 한 예를 나타낸다. 이 실시예는 우선순위 인코더(905), OR 게이트(910), 및 다운스트림(DS) 상태 머신(state machine)(915)을 포함한다. 우선순위 인코더(905)는 CAM의 32비트 출력을 수신한다. CAM이 동일한 포인터를 저장하지 않으면, 32 출력라인 중의 하나만이 텔레포니 신호의 수신시에 액티브하게 된다. 만일 CAM이 32 출력 라인에서 다중 히트를 보고하면(즉, 하나 이상의 라인이 액티브하면), CAM이 부적절하게 프로그램된 것으로 결정되며, 에러 조건이 인터럽트 구조 또는 상태 플래그를 통하여 보고된다.

32개의 출력라인(755) 중의 하나가 액티브할 때, 우선순위 인코더는 액티브 라인의 값에 기초한 5비트 어드레스를 발생시킨다. 이 어드레스는, 제 2 메모리의 발생된 행 어드레스에 저장된 2 제어비트를 검색하기 위해서, CAM으로 다시 보내진다. 이들 2비트는 그후 DS 상태 머신으로 공급된다.

상태 머신은 또한 인코더의 5비트 출력뿐만 아니라 인에이블(enable) 신호(920)를 수신한다. 인에이블 신호는 OR 게이트(910)의 출력이며, OR 게이트는 CAM의 32 출력라인을 입력으로서 수신한다. 이들 라인 중의 하나가 액티브하면, OR 게이트 출력 또한 액티브하다. 이 액티브 신호는 DS 상태 머신으로 하여금 다운스트림 신호 스트림로부터 그 시점에서 수신하고 있는 바이트를 추출하게 한다. 2 제어비트를 디코딩함으로써, DS 상태 머신은 최초에 추출된 바이트를 뒤따르는 다른 3바이트를 추출할 필요가 있는지를 결정한다.

발생된 5비트 어드레스 및 2 제어비트의 값에 기초하여, 상태 머신(915)은 어드레스 및 제어 신호를 발생시키고, 추출된 바이트를 데이터 및 제어 RAM(720 및 725)에 저장한다. 본 발명의 몇몇 실시예에서, CAM에 저장된 포인터와 RAM(720 및 725)(신호 스트림에서의 그의 위치가 저장된 포인터와 매치하는 추출된 바이트를 저장하기 위한)의 저장 위치사이의 관계는 하드 부호화(hard coded)이다. 이 하드 부호화 관계는, 추출된 바이트를 저장하기 위한 RAM(720 및 725)에서 위치에 대한 어드레스 신호를 발생시키기 위해, 다운스트림 상태 머신이 매치된 포인터의 5비트 행 어드레스(925)를 단순히 사용하도록 한다.

본 발명의 몇몇 실시예에서, 메모리 인터페이스 유닛은 하드웨어 설계언어를 사용하도록 설계된다. 그러한 실시예에서, 메모리 인터페이스 유닛의 가능한 설계의 하나가 부록 A에 나타나 있다. 이 설계는 베리로그(Verilog) 하드웨어 설계언어

를 사용한다. 다수의 논리 합성기(시놉시스(Synopsys), 뷰로직(viewlogic), 신플리시티(Synplicity), ACEO, 케스케이드 디자인 오토메이션, 이그젬플러(Exemplar), 및 알테라(Altera)와 같은 것들) 제작자들은 이 코드에서 논리를 게이트로 변환할 수 있고, 이에 의해 메모리 인터페이스 유닛에 대한 게이트 레벨 구현을 얻는다.

메모리 인터페이스 유닛은 텔레포니 데이터신호 또는 텔레포니 제어신호 중 어느 하나를 프레임으로부터 추출한다. 두 신호 형태는 그들이 저장되는 방식에 따라 다르게 취급된다. 제어신호는 제어 RAM(725)에 기록되며, 반면에 데이터신호는 데이터 RAM(720)에 기록된다. 본 발명의 몇몇 실시예에서, 제어 RAM은 128×8, 듀얼 포트 RAM이며, 반면에 데이터 RAM은 "스윙"버퍼("핑퐁" 버퍼)로 작동하는 두개의 32×32, 듀얼 포트 RAM으로 형성된다.

RAM(720 및 725)은 텔레포니 데이터 및 제어 신호가 19MHz, SDH식 시간 영역에서 4MHz, 텔레포니 영역(예를 들면, POTS 영역)으로 크로스(cross)하는 곳이다. 두개의 RAM은 텔레포니 데이터신호를 "스윙"버퍼를 형성하는데 사용된다. "스윙"버퍼는 두개의 메모리 구역을 가지며 따라서 19MHz 영역이 한 프레임에 대한 데이터를 기록할 수 있으며, 반면에 4MHz 영역은 이전의 프레임에 대한 데이터를 판독할 수 있다. 메모리 인터페이스 유닛(715)은 "스윙" RAM 중의 어느 것이 현재의 프레임에서 텔레포니 데이터신호를 저장할지를, 프레임 카운터(도시생략)에 의해 발생된 프레임 카운트에 기초하여 결정한다.

제어신호는 19MHz-4MHz 경계를 크로스하는데 하나 이상의 RAM을 필요로 하지 않는다. 이것은, 제어신호의 주파수가 충분히 느리므로 SDH식 프레임에 다음 시그널링 바이트가 나타나기 전에 19MHz 및 4MHz 영역 모두가 RAM을 액세스할 수 있기 때문이다. 또한, 약간의 부정확성은 제어신호에 대해서는 감수할 만 한데, 그런 부정확성은 사용자에게는 거의 인식되지 않는다. 이 제어신호 RAM의 액세싱에서 4MHz 측이 19MHz 측보다 우선순위를 가진다.

일단 ONU가 완전히 프레임을 수신하면, 카운터(705) 및 CAM(710)은 각 수신된 바이트의 위치를 체크하며, 메모리 인터페이스 유닛(715)은 프레임의 모든 텔레포니 데이터 및 제어 신호를 추출하며 데이터 및 제어 RAM에 저장하며, 포인터 RAM 제어(740), 포인터의 테이블(735), 및 DS TIU 인터페이스(730)는 RAM(720 및 725)의 텔레포니 신호를 적당한 TIU로 보낸다. 즉, 포인터 RAM 제어(740), 포인터의 테이블(735), 및 DS TIU 인터페이스(730)는 어느 데이터 및 제어가 어느 TIU로 보내지는지를 결정하는 교차접속(cross connect) 구조로서 동작한다.

도 7에 도시된 바와 같이, DS TIU 인터페이스(730)는 4개의 TIU A, B, C, 및 D에 연결된다. 또한 이 인터페이스는 포인터 제어(740)에 연결되어서, 판독될 필요가 있는 데이터 및 제어 RAM의 위치 어드레스를 수신한다. 포인터 제어는 이들 어드레스를 포인터의 테이블(735)로부터 검색한다. 프로그램 모드 동안 마이크로프로세서로부터 어드레스 신호를 수신 및 저장하기 위해서, 포인터 테이블은 포인터 제어(740) 및 마이크로프로세서 인터페이스 유닛(745)을 통해 마이크로프로세서(625)에 연결되어 있다.

포인터 제어(740)는 테이블(735)에의 액세스를 제어하는 조정자로서의 역할한다. 포인터 제어는 4.096MHz 클록 신호, 프레임 동기 신호, 및 DS 동기 신호를 수신한다. 이 포인터 제어의 비트 카운터(도시생략)는 4MHz 클록 신호를 수신하여 이 주파수의 9비트 카운트를 발생시킨다. 제어 유닛은 (프레임기(610)에서 발생되는) 프레임 싱크 신호를 수신하여 각 프레임의 정렬을 유지한다. 제어 유닛(740)은, 그의 카운터를 SDH 영역 카운터와 동기화하여 그 결과로 인터페이스 유닛(715 및 730)이 메모리(720 및 725)를 동시에 액세스하지 않도록 하기위해, 다운스트림 동기장치(도시생략)로부터 DS 동기 신호를 수신한다.

포인터 제어의 카운터에 의해 발생된 9비트 카운트의 앞의 6비트는 바이트 카운터로서의 역할을 하여 각 수신 프레임에 대한 64바이트를 카운트한다. 이들 64카운트 값은 64 타임슬롯을 정의하며, 이 기간동안 포인터 제어(740)는 (1) 포인터 테이블(735)로 하여금 RAM(720 및 725)에서 256 어드레스를 발생하도록 하며, 및 (2) 이들 256 어드레스를 DS TIU 인터페이스(730)로 공급한다.

특히, 64 타임슬롯의 각각 동안, 포인터 RAM 제어는 테이블(735)이 4개의 9비트 와이드 기억장소의 내용을 특별한 순서에 따라 출력하도록 한다. 테이블(735)의 각 9비트 와이드 기억장소는, 행 어드레스를 정의하기 위한 7비트, 제어 RAM을 선택하기 위한 1비트, 및 데이터 RAM을 선택하기 위한 1비트로 이루어진다. 바꾸어 말하면, 64 타임슬롯의 각각 동안, TIU 인터페이스(730)는 테이블에 의해 출력된 4 어드레스의 각각에 RAM을 판독한다. 그러면 TIU 인터페이스는 4개의 검색된 신호 세트를 4개의 TIU A, B, C, 및 D로 보낸다.

도 10은 본 발명의 한 실시예에서의 판독 연산을 도식적으로 나타낸 것이며, 120 텔레포니 데이터 바이트가 각 데이터 RAM(720)에 저장되고, 120 텔레포니 제어 바이트가 제어 RAM(725)에 저장된다. 처음 2 타임슬롯 동안 및 마지막 2 타임슬롯 동안, DS TIU 인터페이스는 오버헤드 바이트(예를 들면, 패러티 바이트)를 판독한다.

나머지 60 타임슬롯 동안은, 4개의 30 텔레포니 데이터 바이트 세트(D) 및 4개의 30 텔레포니 제어 바이트 세트(C)가 각각 데이터 RAM(720) 및 제어 RAM(725)로부터 판독된다. 각 TIU(355)는 하나의 텔레포니 데이터 바이트 세트 및 하나의 텔레포니 제어 바이트 세트를 수신한다. 도 10에 도시된 연산에서, 포인터 RAM 제어(740)는 각각의 TIU에 대한 텔레포니 데이터 바이트 및 텔레포니 제어 바이트를 택일적으로 판독한다.

데이터 및 제어 신호의 페칭(fetching)은 특별한 순서에 따라 각각의 TIU에 대해서 이루어진다. 가운데의 60 타임슬롯의 각각 동안, DS TIU 인터페이스(730)는 4 텔레포니 바이트를 검색하며, 세개의 TIU에 대한 처음 3바이트는 중간 레지스터에 저장된다. 4번째 바이트가 패치되었을 때, 4 바이트 모두는 시프트(shift) 레지스터에 기록된다. 그후 이들 바이트는 포인터 제어의 비트 카운터의 값에 기초하여 한번에 한 비트씩 TIU로 시프트 아웃된다. 시프트 레지스터에 있는 바이트가 시프트 아웃되는 동안, 그다음 4 텔레포니 바이트가 페치된다.

본 발명의 몇몇 실시예에서, DS TIU 인터페이스(730) 및 포인터 제어(740)는 하드웨어 설계언어를 사용하여 설계 및 제작된다. 그러한 실시예에서, DS TIU 인터페이스 유닛에 대한 가능한 설계의 한가지가 부록 B에 나타나 있으며, 포인터 제어(740)에 대한 가능한 설계의 하나가 부록 C에 나타나 있다. 이들 설계는 베리로그 하드웨어 설계언어를 사용한다. 다수의 논리 합성기(시놉시스, 뷰로직, 신플리시티, ACEO, 케스케이드 디자인 오토메이션, 이그젬플러, 및 알테라와 같은 것들) 제작자들은 이 코드에서 논리를 게이트로 변환할 수 있고, 이에 의해 DS TIU 인터페이스에 대한 게이트 레벨 구현을 얻는다.

FTTC 네트워크(300)에서 구현되는 본 발명의 또다른 실시예는 텔레포니 애플리케이션 신호 및 넌-텔레포니 애플리케이션 신호를 멀티플렉스하여, FTTC 네트워크를 통해 HDT(320)로 전송하기 위한 통합된 텔레포니/넌-텔레포니 신호 스트림을 형성한다. 이 실시예는, 신호 스트림에서 텔레포니 신호의 적절한 위치를 식별하는 위치식별 지시자(예를 들면, 포인터)를 저장하며, 위에서 언급한 바와 같이, SDH식 프레임에서 구성된다. 그후 전송을 위한 신호를 수신하면 위치식별 지시자를 발생시키며, 이 지시자를 저장된 지시자와 비교한다. 발생된 지시자가 저장된 지시자에 매치하면, 발생된 지시자에 대응하는 텔레포니 신호를 신호 스트림에 삽입한다.

도 11은 본 발명의 그러한 한 실시예를 나타낸다. 이 실시예는 텔레포니 신호 및 넌-텔레포니 신호를 멀티플렉스하여, FTTC 네트워크를 통해 HDT(320)로 전송하기 위한 통합된 텔레포니/넌-텔레포니 신호 스트림을 형성한다. 인터페이스 장치(1100)는 4개의 텔레포니 데이터열(예를 들면, 4개의 DS0열)을 4개의 TIU A, B, C, 및 D로부터 계속해서 수신한다. 이 장치는 이들 열을 또다른 신호 스트림과 멀티플렉스하여, 프레임기에 공급되는 바이트-와이드 열을 형성한다.

도 7의 장치(700)와 같이, 도 11의 장치(1100)는 CCU의 TIUI에서 구현될 수 있는데, TIUI는 TIU와 광섬유 케이블 사이의 인터페이스로서 역할을 한다. 또한, 장치(700)와 같이, 이 장치는 바이트 카운터(1105), CAM(1110), 메모리 인터페이스 유닛(1115), 두개의 듀얼 포트 데이터 RAM(1120), 듀얼 포트 제어 RAM(1125), 업스트림(US) TIU 인터페이스(1130), 테이블(1135), 포인터 제어(1140), 및 마이크로프로세서 인터페이스 유닛(1145)을 가진다. 그러나, 이 장치는 또한 지연 FIFO(1150), 어얼리 카운터(1155), 및 8개의 OR 게이트 세트(1160)를 가진다.

장치(1100)의 동작은 장치(700)의 동작과 유사한데, 역순서라는 것 및 내장된 지연(built in delay)을 가진다는 점이 다르다. 예를 들면, 포인터 제어(740), 테이블(735), 및 DS TIU 인터페이스(730)와 같이, 포인터 제어(1140), 테이블(1135), 및 US TIU 인터페이스(1130)는 교차접속 구조로서 동작한다. 그러나, 장치(700)의 이들 구조와는 달리, 포인터 제어(1140), 테이블(1135), 및 US TIU 인터페이스(1130)는 TIU로부터 유입되는 신호를 RAM(1120 및 1125)로 보낸다.

도 11에 도시된 바와 같이, US TIU 인터페이스(1130)는 TIU A, B, C, 및 D에 연결되어 있어서 텔레포니 데이터 및 제어 바이트를 수신한다. 이 인터페이스는 또한 제어 유닛(1140)을 통해 테이블(1135)에 연결되어 있어서 데이터 및 제어 RAM(1120 및 1125) 내의 위치 어드레스를 수신하며, 이들 RAM 내에 수신된 텔레포니 바이트가 저장된다. 프로그램 모드 동안 마이크로프로세서로부터 어드레스 신호를 수신 및 저장하기 위해서, 테이블(1135)은 제어 유닛(1140) 및 마이크로프로세서 인터페이스 유닛(1145)을 통해서 마이크로프로세서(625)에 연결되어 있다.

테이블(1135)은 비트 카운터(도시생략)를 포함하는 포인터 제어(1140)에 의해 제어된다. 이 제어 유닛은 4.096MHz 클록 신호, 프레임 동기 신호, 및 US 동기 신호를 수신한다. 제어 유닛의 비트 카운터는 4.096MHz 클록 신호를 수신하여 이 주파수로 9비트 카운트를 발생시킨다. 제어 유닛은 (프레임기(610)에서 발생된) 프레임 동기 신호를 수신하여, 각 프레임의 정렬을 유지한다. 제어 유닛(1140)은, 그의 카운터를 SDH 영역 카운터와 동기화하여 그 결과로 인터페이스 유닛(1115 및 1130)이 메모리(1120 및 1125)를 동시에 액세스하지 않도록 하기위해, US 동기 신호를 다운스트림 동기장치(도시생략)로부터 수신한다.

포인터 제어의 카운터에 의해 발생된 9비트 카운트의 앞의 6비트는, 각 전송된 프레임에 대한 64 타임슬롯을 카운트하는 카운터로서의 역할을 한다. 이들 64 타임슬롯 동안 포인터 제어(1140)는, (1) 포인터 테이블(1135)로 하여금 RAM(1120 및 1125)에서 256어드레스를 출력하도록 하며, 및 (2) 이들 어드레스를 US TIU 인터페이스(1130)에 공급한다. 그후 US TIU 인터페이스는 이들 256 어드레스를 사용하여 TIU(355)로부터 256바이트를 RAM(1120 및 1125)에 저장한다.

특히, 64 타임슬롯의 각각 동안, US TIU(1130)는 4개의 TIU A, B, C, 및 D로부터 4 텔레포니 바이트를 수신하여 저장한다. 또한, 각 타임슬롯 동안, 포인터 RAM 제어(1140)는 테이블(1135)이 4개의 9비트 와이드 기억장소를 출력하도록 한다. 테이블(1135)의 각 9비트 와이드 기억장소는 행 어드레스를 정의하는 7비트, 제어 RAM을 선택하는 1비트, 및 데이터 RAM을 선택하는 1비트를 포함한다. US TIU 인터페이스는 테이블의 9비트 출력을 사용하여 RAM(1120 및 1125)에 4 텔레포니 바이트를 저장한다.

상기한 바와 같이, 본 발명의 몇몇 실시예에서, 포인터 RAM 제어(740)는 하드웨어 설계언어를 사용하여 설계 및 제작된다 ; 그러한 설계의 한 예가 부록 C에 나타나 있다. 또한, 본 발명의 몇몇 실시예에서, US TIU 인터페이스(1130) 역시 하드웨어 설계언어를 사용하여 설계 및 제작된다. 그러한 실시예에서, US TIU 인터페이스 설계의 가능한 하나가 부록 D에 나타나 있다. 이 설계는 베리로그 하드웨어 설계언어를 사용한다. 다수의 논리 합성기(시놉시스, 뷰로직, 신플리시티, ACEO, 케스케이드 디자인 오토메이션, 이그젬플러, 및 알테라와 같은 것들) 제작자들은 이 코드에서 논리를 게이트로 변환할 수 있고, 이에 의해 US TIU 인터페이스에 대한 게이트 레벨 구현을 얻는다.

메모리 인터페이스 유닛(1115)은 또한 데이터 및 제어 RAM(1120 및 1125)을 액세스한다. 그러나, 이들 RAM의 4.096MHz 포트에 연결된 US TIU 인터페이스(1130)와는 달리, 메모리 인터페이스 유닛(1115)은 이들 RAM의 19.44MHz 포트에 연결되어 있다. 도 11에 도시된 바와 같이, 메모리 인터페이스 유닛은 또한 CAM(1110), 어얼리 카운터(1155), 카운터(1105), 및 지연 FIFO(1150)에 연결되어 있어서, RAM으로부터 텔레포니 바이트를 수신하며 이들 바이트를 전송을 위한 신호 스트림로 적절한 때에 출력한다.

특히, 어얼리 카운터(1155)는 메모리 인터페이스 유닛으로 하여금 텔레포니 신호를 RAM으로부터 판독 및 이들을 지연 FIFO(1150)에 저장하도록 한다. 어얼리 카운터에 의해 발생된 카운트 값은 카운터(1105)에 의해 발생된 카운트 값에 앞서 미리결정된 바이트 숫자(예를 들면, 24)이다. 어얼리 카운터(1155)는 전송을 위한 프레임에서 장소(또는 타임슬롯 또는 시간주기)를 식별하는 카운트 값을 발생시킨다.

이들 카운트 값은 CAM(1110)에 공급된다. CAM의 한 예가 도 8에 도시되어 있으며, 위에서 설명하였다. CAM(1110)은 32 포인터를 제 1 메모리에 저장하며, 이 메모리에서 각 포인터는 전송을 위한 각 프레임내의 텔레포니 애플리케이션 바이트의 장소를 지정한다. CAM은 또한 각 포인터에 대해 2 제어비트를 제 2 메모리에 저장한다.

일단 CAM이 어얼리 카운터로부터 카운트 값을 수신하면, 카운트 값을 모든 포인터와 동시에 비교한다. 만일 발생된 카운트 값이 CAM의 저장된 포인터 중의 하나에 매치하면, CAM은 매치된 포인터에 대한 출력라인으로 히트 신호를 출력한다.

메모리 인터페이스 유닛(1115)은 CAM의 32 출력라인을 수신한다. CAM이 히트 신호를 출력할 때, (장치(700)의 메모리 인터페이스 유닛(715)과 같은) 메모리 인터페이스 유닛은 우선순위 인코더를 사용하여 32 출력라인 상의 신호를 5비트 출력으로 인코드한다. 이 5비트 출력은 매치된 포인터를 저장한 행의 어드레스를 나타낸다. 이 출력은, 매치된 포인터에 대응하는 제어신호의 쌍을 판독하기 위해, CAM에 피드백된다. 그후 메모리 인터페이스 유닛은 이들 2비트를 수신한다.

본 발명의 몇몇 실시예에서, CAM(710)에 저장된 포인터와 RAM(1120 및 1125)에서의 기억장소 사이의 관계는 하드 부호화이다. 이 하드 부호화 관계는 메모리 인터페이스 유닛으로 하여금 검색된 2 제어비트가 더해진 매치된 포인터의 행 어드레스를 사용하게 하여, RAM(1120 및 1125)로부터 1 내지 4 바이트를 추출하게 한다. 본 발명의 몇몇 실시예에서, 메모리 인터페이스 유닛(1115)은 하드웨어 설계언어를 사용하여 설계 및 제작된다. 그러한 실시예에서, 하나의 가능한 메모리 인터페이스 유닛의 설계는 부록 E에 나타나 있다. 이 설계는 베리로그 하드웨어 설계언어를 사용한다. 다수의 논리 합성기(시놉시스, 뷰로직, 신플리시티, ACEO, 케스케이드 디자인 오토메이션, 이그젬플러, 및 알테라와 같은 것들) 제작자들은 이 코드에서 논리를 게이트로 변환할 수 있고, 이에 의해 메모리 인터페이스 유닛(1115)에 대한 게이트 레벨 구현을 얻는다.

메모리 인터페이스 유닛은 추출된 텔레포니 데이터 바이트를 지연 FIFO(1150)에 저장하며, 반면에 추출된 텔레포니 제어 바이트의 일부를 순환버퍼(circular buffer)에 저장한다. 상기한 바와같이, 어얼리 카운터는 바이트 카운터(1105)에 앞서 미리 결정된 바이트 숫자(예를 들면, 24)이다. 따라서, 어얼리 카운터는 메모리 인터페이스 유닛으로 하여금, 바이트를 출력할 필요가 있기 다수의 사이클 전에 각 텔레포니 바이트를 추출하도록 한다.

메모리 인터페이스 유닛은 지연 FIFO(1150)를 사용하여, 추출된 텔레포니 데이터 바이트를 저장한다. 지연 FIFO는 32개의 9비트 세트를 저장한다. 메모리 인터페이스 유닛은 텔레포니 데이터 바이트를 각 세트의 8비트에 저장할 수 있다. 9번째 비트는 세트가 텔레포니 데이터 바이트를 포함하고 있는지를 표시한다. 어얼리 카운터가 카운트 값을 발생시키는 각 시간에, 메모리 인터페이스 유닛은 (1) 먼저 입력된 비트 세트를 판독하며, 및 (2) 텔레포니 데이터 바이트를 FIFO에 저장한다. 또한, 어얼리 카운터가 새로운 카운트 값을 발생시키는 각 시간에, FIFO 내의 9비트 레지스터의 각각에 있는 내용은 한 단계 전진된다.

어얼리 카운트 및 지연 FIFO는, 몇 클록 사이클 후에 드러날 텔레포니 데이터 바이트를 식별하기 위해, 메모리 인터페이스 유닛으로 하여금 "미리보기(look ahead)" 기능을 실행하도록 한다. 근소한 시간만큼 앞서서 텔레포니 데이터 바이트를 식별함으로써, 메모리 인터페이스 유닛은, 도 5에 도시된 것과 같이, 텔레포니 제어 바이트 및 텔레포니 데이터 바이트를 적절한 순서로 결합할 수 있다. 즉, 이 어얼리 히트 검출은 적절한 시기에 데이터 및 제어 바이트가 페치되어 SDH식 프레임에 합쳐지도록 한다.

메모리 인터페이스 유닛은 순환버퍼에 있는 추출된 텔레포니 제어 바이트를 저장한다. 본 발명의 몇몇 실시예에서, 3바이트 순환버퍼가 메모리 인터페이스 유닛에 의해 사용된다. 이들 실시예에서는, 이 유닛은 프레임 카운터(도시생략)에서 발생된 프레임 카운트를 사용하여, 9바이트 그룹의 전송된 셀에 있는 8 텔레포니 데이터 바이트의 세트 앞에 위치할 수 있는 가능성있는 8 텔레포니 제어 바이트 중의 하나를 선택적으로 순환버퍼에 저장한다.

메모리 인터페이스 유닛은 어얼리 카운터(1155) 또는 카운터(1105)의 제어하에서 지연 RAM 및 순환버퍼의 내용을 판독한다. 메모리 인터페이스는 각각의 검색된 바이트를 OR 게이트(1160)의 입력에 공급한다. OR 게이트는 메모리 인터페이스 유닛의 출력 또는 또다른 신호 스트림을 수신하지만, 이 둘을 동시에 수신하지는 않는다. 그후 이 게이트는 프레임기(610)에 연결된 라인(1165) 상으로 수신한 신호를 출력한다.

상기한 것으로부터 명백하듯이, 신호 스트림로부터 (텔레포니 애플리케이션 및 넌-텔레포니 애플리케이션과 같은) 서로 다른 통신애플리케이션에 대한 신호를 효과적으로 디멀티플렉스하는 방법 및 장치를 본 발명이 제공하기 때문에, 본 발명의 이점이 있다. 또한 본 발명은 서로 다른 애플리케이션에 대한 신호를 신호 스트림로 효과적으로 멀티플렉스하는 방법 및 장치를 제공한다.

상기한 바와 같이, 본 발명의 몇몇 실시예는 CAM을 사용하여 디멀티플렉싱 동작을 효과적으로 수행하여, 신호 세트가 특별한 형태의 통신애플리케이션에 대한 것인지를 신속히 검사한다. 만일 그렇다면, 이들 실시예는 이들 신호 세트를 신호 스트림로부터 추출한다. 이 디멀티플렉싱 동작은, 어느 신호 세트가 특별한 통신애플리케이션으로 보내져야 하는지를 결정하기 이전에, 수신된 모든 신호 세트를 저장해야 할 필요성을 제거하였다. 본 발명의 몇몇 실시예는 또한 CAM을 사용하여 멀티플렉싱 동작을 효과적으로 수행하여, 신호 스트림에서 적절한 위치를 신속히 식별한다.

비록 본 발명이 다수의 특정한 설명을 참조하여 기술되었지만, 본 발명의 범주를 벗어나지 않고도 다른 특정 형식으로 본 발명이 실시될 수 있음을 당업자는 알 것이다. 따라서, 당업자는 본 발명이 앞에서 설명된 것에 의해 제한되어서는 안된다는 것을 알 것이며, 본 발명은 첨부되는 청구항에 의해서 정의된다.

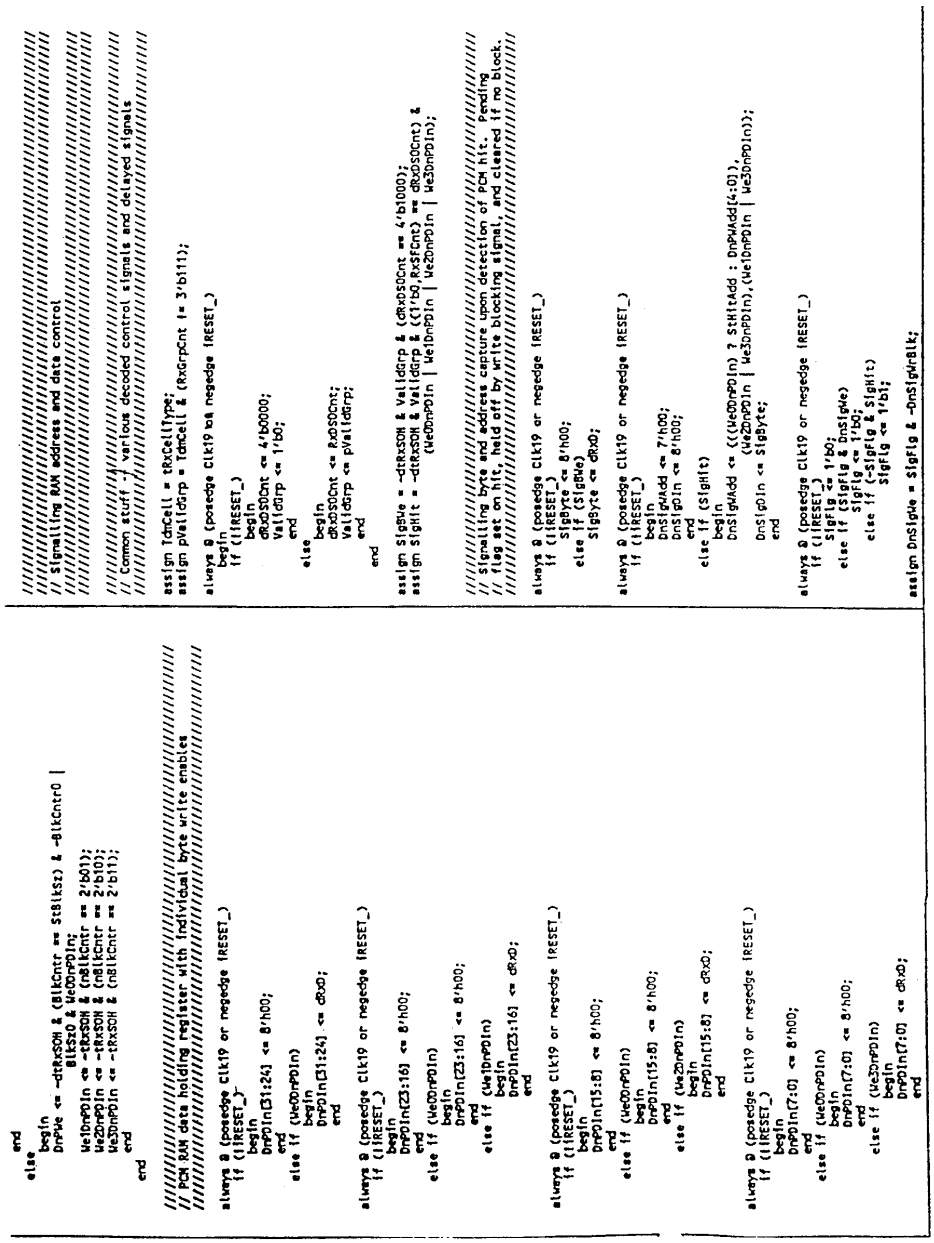
부록 A

```
//
// tdsrmictl.v :   Down-stream RAM input control
//
//
//
//
//
// functional desription:
//
//
//
// synthesis constraints
//
// sc   ClkList = {Clk19}
// sc   create_clock Clk19 -period 40 -waveform {0 20}
// sc   set_clock_skew -uncertainty 3 ClkList
// sc   set_fix_hold ClkList
// sc   set_false_path -from {iRESET_}
// sc   set_dont_touch_network ClkList
// sc   set_load 2 DsClk19b

module tdsrmictl
    (
    iRESET_,
    Clk19,
    DsClk19b,
    tRxSOH,
    tRxCellType,
    RxD,
    RxSFCnt,
    RxGrpCnt,
    RxDSOCnt,
    RxPcmHit,
    StBlkSz0,
    StBlkSz1,
    DnPDIn,
    DnPWAdd,
    DnPWe,
    DnSigDIn,
    DnSigWAdd,
    DnSigWe,
    DnSigWrBlk,
    CAMErr
    );

input           iRESET_;
input           Clk19;
input   [7:0]   RxD;            //Down-stream SONET frame data
input           tRxCellType;    //Down-stream cell type
input           tRxSOH;         //Down-stream SONET overhead enable
input   [2:0]   RxSFCnt;        //Down-stream super frame count
input   [2:0]   RxGrpCnt;       //POTS group number within the current
                                //down-stream TDM cell
input   [3:0]   RxDSOCnt;       //DSO number within the current
                                //down-stream POTS group
input   [31:0]  RxPcmHit;       //down-stream hit signals from CAM
input   [31:0]  StBlkSz0;       //Bit 0 of all stored CAM block size counts
input   [31:0]  StBlkSz1;       //Bit 1 of all stored CAM block size counts
input           DnSigWrBlk;     //

output          DsClk19b;       //
output  [31:0]  DnPDIn;         //Down-stream PCM RAM Data-in
output  [5:0]   DnPWAdd;        //Down-stream PCM RAM Address-in
output          DnPWe;          //Down-stream PCM RAM Write Enable
output  [7:0]   DnSigDIn;       //Down-stream Signalling RAM Data-in
output  [6:0]   DnSigWAdd;      //Down-stream Signalling RAM Wr Address
output          DnSigWe;        //Down-stream Signalling RAM Wr Enable
output          CAMErr;         //

// local declarations

wire    [1:0]   BlkSz;          //
wire    [31:0]  RxPcmHit;       //
wire            BlkCntr0;       //
wire            BlkSz0;         //
wire            Busy;           //
wire            ErrOvrLp;       //
reg             We0DnPDIn;      //
reg             We1DnPDIn;      //
reg             We2DnPDIn;      //
reg             We3DnPDIn;      //
reg     [31:0]  DnPDIn;         //
reg     [4:0]   HitAdd;         //
reg             Hit;            //
reg             ErrMulHitFlg;   //
reg             ErrMulHit;      //
reg     [4:0]   StHitAdd;       //
reg     [5:0]   DnPWAdd;        //
reg     [1:0]   StBlkSz;        //
reg             dtRxSOH;        //
reg     [7:0]   dRxD;           //
reg     [1:0]   BlkCntr;        //
reg     [1:0]   nBlkCntr;       //
reg             DnPWe;          //
reg             ValidGrp;       //
reg             SigFlg;         //
integer         i;              //
reg     [3:0]   dRxDSOCnt;      //
reg     [7:0]   SigByte;        //
reg     [7:0]   DnSigDIn;       //
reg     [6:0]   DnSigWAdd;      //
reg             CAMErr;         //

wire            TdmCell;        //
wire            pValidGrp;      //
wire            SigHit;         //
wire            Sig8We;         //
wire            DnSigWe;        //
wire            DsClk19b;       //

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//// Common stuff
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Clock inversion for down-stream signalling and PCM RAMs
////////////////////////////////////////////////////////////////////////////////

assign DsClk19b = ~Clk19;

////////////////////////////////////////////////////////////////////////////////
// Priority encoder with error signal for more than one CAM location hit
////////////////////////////////////////////////////////////////////////////////

always @ ( RxPcmHit )
        begin
            Hit = 1'b0;
            HitAdd = 5'b00000;
            ErrMulHit = 1'b0;
            ErrMulHitFlg = 1'b0;
```

```
            for (i = 31; i >= 0; i = i -1)
                begin
                if (RxPcmHit[i])
                    begin
                    Hit = 1'b1;
                    if (ErrMulHitFlg) ErrMulHit = 1'b1;
                    ErrMulHitFlg = 1'b1;
                    HitAdd = i;
                    end
                end
        end

assign ErrOvrLp = Busy & Hit;

always @ (posedge Clk19 or negedge iRESET_)
    begin
    if (!iRESET_)
        begin
        CAMErr <= 1'b0;
        end
    else
        begin
        CAMErr <= ErrMulHit | ErrOvrLp;
        end
    end

////////////////////////////////////////////////////////////////////////////////
// Hit address latch            Latches address on CAM hit
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk19 or negedge iRESET_)
    begin
    if (!iRESET_)
        begin
        StHitAdd <= 5'b00000;
        end
    else if (Hit & ~Busy)
            begin
            StHitAdd <= HitAdd;
            end
    end

////////////////////////////////////////////////////////////////////////////////
// RAM address latch
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk19 or negedge iRESET_)
    begin
    if (!iRESET_)
        begin
        DnPWAdd <= 6'b000000;
        StBlkSz <= 2'b00;
        end
    else if (We0DnPDIn)
            begin
            DnPWAdd <= {RxSFCnt[0],StHitAdd};
            StBlkSz <= BlkSz;
            end
    end

////////////////////////////////////////////////////////////////////////////////
// Block size multiplexer to retrive the stored block size of the hit CAM
// location
////////////////////////////////////////////////////////////////////////////////

assign BlkSz = {StBlkSz1[StHitAdd],StBlkSz0[StHitAdd]};
```

```
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//// PCM RAM address and data control
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Delayed Hit signal used as byte 0 write enable, counter start signal
// and RAM storage enable
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk19 or negedge iRESET_)
    begin
    if (!iRESET_)
        begin
        We0DnPDIn <= 1'b0;
        dtRxSOH <= 1'b0;
        dRx0 <= 8'h00;
        end
    else
        begin
        We0DnPDIn <= Hit & ~Busy;
        dtRxSOH <= tRxSOH;
        dRx0 <= Rx0;
        end
    end

////////////////////////////////////////////////////////////////////////////////
// Byte counter and control
////////////////////////////////////////////////////////////////////////////////

assign BlkCntr0 = (BlkCntr == 2'b00);
assign BlkSz0 = (BlkSz == 2'b00);

always @ (posedge Clk19 or negedge iRESET_)
    begin
    if (!iRESET_)
        BlkCntr <= 2'b00;
    else
        BlkCntr <= nBlkCntr;
    end

always @ (We0DnPDIn or StBlkSz or BlkSz0 or dtRxSOH or BlkCntr0 or BlkCntr)
    begin
    if (We0DnPDIn & ~BlkSz0)
        nBlkCntr = 2'b01;
    else if (!dtRxSOH)
            if ((BlkCntr >= StBlkSz) || BlkCntr0)
                nBlkCntr = 2'b00;
            else
                nBlkCntr = BlkCntr + 1'b1;
        else
            nBlkCntr = BlkCntr;
    end

assign Busy = nBlkCntr != 2'b00;

always @ (posedge Clk19 or negedge iRESET_)
    begin
    if (!iRESET_)
        begin
        DnPWe <= 1'b0;
        We1DnPDIn <= 1'b0;
        We2DnPDIn <= 1'b0;
        We3DnPDIn <= 1'b0;
```


등록특허 10-0507023



```
    end
  else
    begin
    DnPWe <= ~dtRxSOH & (BlkCntr == StBlkSz) & ~BlkCntr0 |
             BlkSz0 & WeODnPDin;
    We1DnPDin <= ~dtRxSOH & (nBlkCntr == 2'b01);
    We2DnPDin <= ~dtRxSOH & (nBlkCntr == 2'b10);
    We3DnPDin <= ~dtRxSOH & (nBlkCntr == 2'b11);
    end
end

////////////////////////////////////////////////////////////////////////////////
// PCN RAM data holding register with individual byte write enables
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk19 or negedge !RESET_)
  if (!!RESET_)
    begin
    DnPDIn[31:24] <= 8'h00;
    end
  else if (WeODnPDin)
    begin
    DnPDIn[31:24] <= dRxD;
    end

always @ (posedge Clk19 or negedge !RESET_)
  if (!!RESET_)
    begin
    DnPDIn[23:16] <= 8'h00;
    end
  else if (We1DnPDin)
    begin
    DnPDIn[23:16] <= dRxD;
    end

always @ (posedge Clk19 or negedge !RESET_)
  if (!!RESET_)
    begin
    DnPDIn[15:8] <= 8'h00;
    end
  else if (We2DnPDin)
    begin
    DnPDIn[15:8] <= dRxD;
    end

always @ (posedge Clk19 or negedge !RESET_)
  if (!!RESET_)
    begin
    DnPDIn[7:0] <= 8'h00;
    end
  else if (We3DnPDin)
    begin
    DnPDIn[7:0] <= dRxD;
    end

////////////////////////////////////////////////////////////////////////////////
// various decoded control signals and delayed signals
////////////////////////////////////////////////////////////////////////////////

assign TdCell = tRxCellType;
assign pValidDrp = TdCell & (RxGrpCnt != 3'b111);

always @ (posedge Clk19 or negedge !RESET_)
  begin
  if (!!RESET_)
    begin
    dRxDSOCnt <= 4'b0000;
    ValidDrp <= 1'b0;
    end
  else
    begin
    dRxDSOCnt <= RxDSOCnt;
    ValidDrp <= pValidDrp;
    end
  end

assign SigWe = ~dtRxSOH & ValidDrp & (dRxDSOCnt == 4'b1000);
assign SigHit = ~dtRxSOH & ValidDrp & ((1'b0,RxSFCnt) == dRxDSOCnt) &
                (WeODnPDin | We1DnPDin | We2DnPDin | We3DnPDin);

////////////////////////////////////////////////////////////////////////////////
// signalling byte and address capture upon detection of PCN hit.  Pending
// flag set on hit, held off by write blocking signal, and cleared if no block.
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk19 or negedge !RESET_)
  if (!!RESET_)
    SigByte <= 8'h00;
  else if (SigWe)
    SigByte <= dRxD;

always @ (posedge Clk19 or negedge !RESET_)
  if (!!RESET_)
    begin
    DnSigWAdd <= 7'h00;
    DnSigDIn <= 8'h00;
    end
  else if (SigHit)
    begin
    DnSigWAdd <= {{(WeODnPDin) ? StHitAdd : DnPWAdd[4:0]},
                  (We2DnPDin | We3DnPDin),(We1DnPDin | We3DnPDin)};
    DnSigDIn <= SigByte;
    end

always @ (posedge Clk19 or negedge !RESET_)
  if (!!RESET_)
    SigFlg <= 1'b0;
  else if (SigFlg & DnSigWe)
    SigFlg <= 1'b0;
  else if (~SigFlg & SigHit)
    SigFlg <= 1'b1;

assign DnSigWe = SigFlg & ~DnSigWrBlk;
```

부록 B

```
// tdstint.v :  Down-stream telephony interface
//
//
//
//
// Functional description:
//
//
// synthesis constraints
//
// sc  ClkList = {Clk4}
// sc  create_clock Clk4 -period 220 -waveform {0 110}
// sc  set_clock_skew -uncertainty 3 ClkList
// sc  set_fix_hold ClkList
// sc  set_false_path -from {RESET4M_}
// sc  set_dont_touch_network ClkList
// sc  set_dont_touch_network RESET4M
// sc  set_load 32 * load_of(ms/CMOP00/D) DsClk4b
// sc  set_input_delay 1 {DnFrmCnt}

module tdstint
    (
    RESET4M_,
    Clk4,
    DsClk4b,
    DnCntrRst,
    DnFrmRef,
    DnFrmCnt,
    RxAlSL4M,
    CritPwrAlm4M,
    UsPrtyErr,
    UsOffSetRef,
    DnPDOut,
    DnPRAdd,
    DnPREn,
    DnSigDOut,
    DnSigRAdd,
    DnSigREn,
    PtrRAdDS,
    PtrRDOutDUS,
    PtrREnDS,
    DnMsgFC,
    oTCLK,
    oTFS,
    oTDDA,
    oTDDB,
    oTDDC,
    oTDDD,
    UpReq4M,
    UpRdyDaTint4M,
    !UPA,
    !UPRNW,
    !UPD,
    UpRDDaTint,
    BitOffSetChg
    );

input       RESET4M_;
input       Clk4;
input       DnCntrRst;          //
input [4:0] DnFrmCnt;           //
input       RxAlSL4M;           //
input       CritPwrAlm4M;       //
input [3:0]  UsPrtyErr;         //
input [3:0]  UsOffSetRef;       //
input [7:0]  DnPDOut;           //Down-stream PCH RAM Data-Out
input [8:0]  DnSigDOut;         //Down-stream signalling RAM Data-Out
input [8:0]  PtrRDOutDUS;       //
input        UpReq4M;           //
input [10:1] !UPA;              //
input        !UPRNW;            //
input [15:0] !UPD;              //
output       DsClk4b;           //
output [1:0] DnMsgFC;           //Down-stream PCH RAM Read Address-In
output [5:0] DnPRAdd;           //Down-stream PCH RAM Read Enable
output       DnPREn;            //Down-stream frame reference pulse
output       DnFrmRef;          //Down-stream signalling RAM Read Address-In
output [6:0] DnSigRAdd;         //Down-stream signalling RAM Read Enable
output       DnSigREn;          //
output [6:0] PtrRAdDS;          //
output       PtrREnDS;          //Telephony clock
output       oTCLK;             //Telephony frame sync
output       oTFS;              //Serial output data for telephony interface A
output       oTDDA;             //Serial output data for telephony interface B
output       oTDDB;             //Serial output data for telephony interface C
output       oTDDC;             //Serial output data for telephony interface D
output       oTDDD;             //
output       UpRdyDaTint4M;     //
output [15:0] UpRDDaTint;       //
output       BitOffSetChg;      //

// local declarations

reg [8:0]  BitCnt;              //
reg [8:0]  nBitCnt;             //
reg [4:0]  FrmCnt;              //
reg        oTFS;                //
reg        DnFrmRef;            //
reg [7:0]  PCMAldReg4;          //
reg [7:0]  PCMAldReg8;          //
reg [7:0]  PCMAldReg0;          //
reg [6:0]  DnPRAdLwr;           //
reg        SigEn;               //
reg        PCMEn;               //
reg        PtrREnDS;            //
reg [7:0]  SigAldReg4;          //
reg [7:0]  SigAldReg8;          //
reg [7:0]  SigAldReg0;          //
reg [7:0]  TDDAReg;             //
reg [7:0]  TDDBReg;             //
reg [7:0]  TDDCReg;             //
reg [7:0]  TDDDReg;             //
reg [7:0]  PrtyByte4;           //
reg [7:0]  PrtyByte8;           //
reg [7:0]  PrtyByteC;           //
reg [7:0]  PrtyByte0;           //
reg [7:0]  PrtyAccA;            //
reg [7:0]  PrtyAccB;            //
reg [7:0]  PrtyAccC;            //
reg [7:0]  PrtyAccD;            //
reg        pUpRdyDaTint4M;      //
reg [7:0]  MsgCmd;              //
reg [15:0] MsgAdd;              //
reg [7:0]  MsgWrD;              //
reg [7:0]  MsgNmEn;             //
reg [7:0]  PCMSigOff8;          //
reg [7:0]  PCMAlSByte;          //
```

```
assign RsrvByte = 8'd0;

////////////////////////////////////////////////////////////////////////////////
// Clock inversion for down-stream signalling and PCM RAMs
////////////////////////////////////////////////////////////////////////////////

assign DsClk4b = ~Clk4;

////////////////////////////////////////////////////////////////////////////////
// Processor address decodes
////////////////////////////////////////////////////////////////////////////////

assign dAny         = UpReq4M & (!UPA[10:6] == 5'b000_00);
assign dBitOffSet   = dAny & (!UPA[5:1] == 5'b00_000);
assign dPCMSigOffB  = dAny & (!UPA[5:1] == 5'b00_001);
assign dPCMAISByte  = dAny & (!UPA[5:1] == 5'b00_010);
assign dSigAISByte  = dAny & (!UPA[5:1] == 5'b00_011);
assign dAutoAIS     = dAny & (!UPA[5:1] == 5'b00_100);
assign dUpAIS       = dAny & (!UPA[5:1] == 5'b00_101);
assign dMsgXmtEn    = dAny & (!UPA[5:1] == 5'b01_000);
assign dMsgPend     = dAny & (!UPA[5:1] == 5'b01_001);
assign dMsgCmd      = dAny & (!UPA[5:1] == 5'b01_010);
assign dMsgAdd      = dAny & (!UPA[5:1] == 5'b01_011);
assign dMsgWrD      = dAny & (!UPA[5:1] == 5'b01_100);

////////////////////////////////////////////////////////////////////////////////
// Offset bit-count capture
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        BitOffSet <= 9'd0;
        end
    else if (UsOffSetRef)
        begin
        BitOffSet <= BitCnt;
        end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        BitOffSetS <= 9'd0;
        BitOffSetChg <= 1'b0;
        end
    else
        begin
        BitOffSetS <= BitOffSet;
        BitOffSetChg <= BitOffSetS != BitOffSet;
        end
    end

////////////////////////////////////////////////////////////////////////////////
// Porcessor interface control logic, storage regs, and read mux
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        pUpRdyDsTint4M <= 1'b0;
        end
    else
        begin
        pUpRdyDsTint4M <= UpReq4M;
        end
    end

assign UpRdyDsTint4M = dAny & UpReq4M & ~pUpRdyDsTint4M;
assign UpWr      = UpRdyDsTint4M & ~iUPRNW;

assign MsgCmdWr = UpWr & dMsgCmd;

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        MsgCmd <= 8'h00;
        end
    else if (MsgCmdWr)
        begin
        MsgCmd <= iUPD[7:0];
        end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        MsgAdd <= 16'h0000;
        end
    else if (UpWr & dMsgAdd)
        begin
        MsgAdd <= iUPD;
        end
    end

assign MsgHAdd = MsgAdd[15:8];
assign MsgLAdd = MsgAdd[7:0];

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        MsgWrD <= 8'h00;
        end
    else if (UpWr & dMsgWrD)
        begin
        MsgWrD <= iUPD[7:0];
        end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        MsgXmtEn <= 4'h0;
        end
    else if (UpWr & dMsgXmtEn)
        begin
        MsgXmtEn <= iUPD[3:0];
        end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
```


등록특허 10-0507023

```
                                                ((7'd0,BitOffSet)   & (16{dBitOffSet})));

    PCMSigOffB <= 8'h02;
    end
  else if (UpWr & dPCMSigOffB)
    begin
    PCMSigOffB <= IUPD[7:0];
    end

  end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
      begin
      PCMAISByte <= 8'hff;
      end
    else if (UpWr & dPCMAISByte)
      begin
      PCMAISByte <= IUPD[7:0];
      end

    end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
      begin
      SigAISByte <= 8'h02;
      end
    else if (UpWr & dSigAISByte)
      begin
      SigAISByte <= IUPD[7:0];
      end

    end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
      begin
      AutoAIS <= 1'b0;
      end
    else if (UpWr & dAutoAIS)
      begin
      AutoAIS <= IUPD[0];
      end

    end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
      begin
      UpAIS <= 4'd0;
      end
    else if (UpWr & dUpAIS)
      begin
      UpAIS <= IUPD[3:0];
      end

    end

  assign UpRDDataInt = ({8'h00,MsgCmd}      & {16{dMsgCmd}})
                     | ({8'h00,MsgAdd}      & {16{dMsgAdd}})
                     | ({8'h00,MsgWrD}      & {16{dMsgWrD}})
                     | ({12'd0,MsgXmtEn}    & {16{dMsgXmtEn}})
                     | ({15'd0,MsgPend}     & {16{dMsgPend}})
                     | ({8'h00,PCMSigOffB}  & {16{dPCMSigOffB}})
                     | ({8'h00,PCMAISByte}  & {16{dPCMAISByte}})
                     | ({8'h00,SigAISByte}  & {16{dSigAISByte}})
                     | ({15'd0,AutoAIS}     & {16{dAutoAIS}})
                     | ({12'd0,UpAIS}       & {16{dUpAIS}});

//////////////////////////////////////////////////////////////////////////////////
// Master bit counter, reference frame pulse and frame count register
//////////////////////////////////////////////////////////////////////////////////

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
      begin
      BitCnt <= 9'h00;
      DnFrmRef <= 1'b0;
      end
    else
      begin
      BitCnt <= nBitCnt;
      DnFrmRef <= (BitCnt == 9'b11101_0000);
      end

    end

  always @ (BitCnt or DnCntrRst)
    begin
    if (DnCntrRst)
      nBitCnt = 9'h00;
    else
      nBitCnt = BitCnt + 1'b1;

    end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
      FrmCnt <= 5'b00000;
    else if (DnCntrRst | (BitCnt == 9'd511))
      FrmCnt <= DnFrmCnt;
    end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
      DnMsgFC <= 2'b00;
    else if (BitCnt == cUseMsgFCRef)
      DnMsgFC <= FrmCnt[1:0];
    end

//////////////////////////////////////////////////////////////////////////////////
// Address and read enable for pointer and PCM RAMs
//////////////////////////////////////////////////////////////////////////////////

  assign fSSel = nBitCnt[8:3] == 6'd1;

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
      oTFS <= 1'b1;
    else
      oTFS <= (fSSel) ?
                ((FrmCnt ==5'b00000) ?
                    cSFS[7-nBitCnt[2:0]] : cFS[7-nBitCnt[2:0]]) :
                1'b1;

    end

//////////////////////////////////////////////////////////////////////////////////
// Address and read enable for pointer and PCM RAMs
//////////////////////////////////////////////////////////////////////////////////

  assign PtrRAddS = {BitCnt[1:0],BitCnt[8:4]};
```

```
assign pPtrRRenDS =  BitCnt[3:0] == 4'b0110 |
                     BitCnt[3:0] == 4'b1000 |
                     BitCnt[3:0] == 4'b1011 |
                     BitCnt[3:0] == 4'b1101;

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        PtrRRenDS <= 1'b0;
    else
        PtrRRenDS <= pPtrRRenDS;

assign DnPRAdd = {FrmCnt[0],DnPRAddLwr[6:2]};

assign DnPRen =      BitCnt[3:0] == 4'b1000 |
                     BitCnt[3:0] == 4'b1010 |
                     BitCnt[3:0] == 4'b1101 |
                     BitCnt[3:0] == 4'b1111;

assign DnSigRAdd = DnPRAddLwr;

assign DnSigRen   = DnPRen;

////////////////////////////////////////////////////////////////////////////////
// Stored info from pointer RAM
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        DnPRAddLwr <= 7'h00;
        SigEn <= 1'b0;
        PCMEn <= 1'b0;
        end
    else if (PtrRRenDS)
            begin
            DnPRAddLwr <= PtrRDOutDUS[8:2];
            SigEn <= PtrRDOutDUS[1];
            PCMEn <= PtrRDOutDUS[0];
            end
    end

////////////////////////////////////////////////////////////////////////////////
// Data read muxes for PCM and signalling RAMs
////////////////////////////////////////////////////////////////////////////////

assign DnPDOutB  =  (-PCMEn) ? PCMSigOffB : (
                    (DnPDOut[31:24] & {8{DnPRAddLwr[1:0] == 2'b00}}) |
                    (DnPDOut[23:16] & {8{DnPRAddLwr[1:0] == 2'b01}}) |
                    (DnPDOut[15:8] & {8{DnPRAddLwr[1:0] == 2'b10}}) |
                    (DnPDOut[7:0] & {8{DnPRAddLwr[1:0] == 2'b11}})
                    );

assign DnSigOutB  =  (-SigEn) ? PCMSigOffB : DnSigDOut;

////////////////////////////////////////////////////////////////////////////////
// Load enables for holding registers
////////////////////////////////////////////////////////////////////////////////

assign LdTDDA =  BitCnt[3:0] == 4'b1101;
assign LdTDDB =  BitCnt[3:0] == 4'b1010;
assign LdTDDC =  BitCnt[3:0] == 4'b1111;
assign LdTDDD =  BitCnt[3:0] == 4'b1000;

////////////////////////////////////////////////////////////////////////////////
```

```
// Holding registers for PCM and signalling
//
//  PCM holding reg for interface C is not needed because the PCM buffer
//  read occurs just as the shift byte is needed thus no holding reg is
//  required.
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        PCMHldRegA <= 8'h00;
        SigHldRegA <= 8'h00;
        end
    else if (LdTDDA)
            begin
            PCMHldRegA <= DnPDOutB;
            SigHldRegA <= DnSigOutB;
            end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        PCMHldRegB <= 8'h00;
        SigHldRegB <= 8'h00;
        end
    else if (LdTDDB)
            begin
            PCMHldRegB <= DnPDOutB;
            SigHldRegB <= DnSigOutB;
            end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        SigHldRegC <= 8'h00;
        end
    else if (LdTDDC)
            begin
            SigHldRegC <= DnSigOutB;
            end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        PCMHldRegD <= 8'h00;
        SigHldRegD <= 8'h00;
        end
    else if (LdTDDD)
            begin
            PCMHldRegD <= DnPDOutB;
            SigHldRegD <= DnSigOutB;
            end
    end

////////////////////////////////////////////////////////////////////////////////
// Parity generation and storage
////////////////////////////////////////////////////////////////////////////////

assign LdPrty = RsrvBSel & LdShftRegs;
```


등록특허 10-0507023

```
always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        PrtyByteA <= 8'h00;
        PrtyByteB <= 8'h00;
        PrtyByteC <= 8'h00;
        PrtyByteD <= 8'h00;
        PrtyAccA <= 8'h00;
        PrtyAccB <= 8'h00;
        PrtyAccC <= 8'h00;
        PrtyAccD <= 8'h00;
        end
    else if (LdPrty)
            begin
            PrtyByteA <= PrtyAccA;
            PrtyByteB <= PrtyAccB;
            PrtyByteC <= PrtyAccC;
            PrtyByteD <= PrtyAccD;
            PrtyAccA <= ~ShftOByteA;
            PrtyAccB <= ~ShftOByteB;
            PrtyAccC <= ~ShftOByteC;
            PrtyAccD <= ~ShftOByteD;
            end
        else if (LdShftRegs)
            begin
            PrtyAccA <= PrtyAccA ^ ShftOByteA;
            PrtyAccB <= PrtyAccB ^ ShftOByteB;
            PrtyAccC <= PrtyAccC ^ ShftOByteC;
            PrtyAccD <= PrtyAccD ^ ShftOByteD;
            end
    end

////////////////////////////////////////////////////////////////////////////
// Message cpu holding reg, output holding reg, and control logic
////////////////////////////////////////////////////////////////////////////


always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        MsgPend <= 1'b0;
        end
    else if (MsgCmdWr)
            begin
            MsgPend <= 1'b1;
            end
        else if (MsgHldLd)
                begin
                MsgPend <= 1'b0;
                end

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        MsgEnA <= 1'b0;
        MsgEnB <= 1'b0;
        MsgEnC <= 1'b0;
        MsgEnD <= 1'b0;
        end
    else if (MsgHldWrDClr)
            begin
            MsgEnA <= MsgXmtEn[0];
            MsgEnB <= MsgXmtEn[1];
            MsgEnC <= MsgXmtEn[2];
            MsgEnD <= MsgXmtEn[3];
            end

assign MsgHldLd = MsgPend & MsgHldWrDClr;

assign MsgHldCmdClr  = LdShftRegs & MsgBCmdSel;
assign MsgHldHAddClr = LdShftRegs & MsgBHAddSel;
assign MsgHldLAddClr = LdShftRegs & MsgBLAddSel;
assign MsgHldWrDClr  = LdShftRegs & MsgBWrDSel;

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        MsgHldCmd <= 8'h00;
        end
    else if (MsgHldLd)
            begin
            MsgHldCmd <= MsgCmd;
            end
        else if (MsgHldCmdClr)
                begin
                MsgHldCmd <= 8'h00;
                end

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        MsgHldHAdd <= 8'h00;
        end
    else if (MsgHldLd)
            begin
            MsgHldHAdd <= MsgHAdd;
            end
        else if (MsgHldHAddClr)
                begin
                MsgHldHAdd <= 8'h00;
                end

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        MsgHldLAdd <= 8'h00;
        end
    else if (MsgHldLd)
            begin
            MsgHldLAdd <= MsgLAdd;
            end
        else if (MsgHldLAddClr)
                begin
                MsgHldLAdd <= 8'h00;
                end

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        MsgHldWrD <= 8'h00;
        end
    else if (MsgHldLd)
            begin
            MsgHldWrD <= MsgWrD;
            end
        else if (MsgHldWrDClr)
                begin
                MsgHldWrD <= 8'h00;
                end
```


등록특허 10-0507023

```
////////////////////////////////////////////////////////////////////////////////
// Alarms
////////////////////////////////////////////////////////////////////////////////

assign AISA = AutoAIS & RxAISL4M | UpAIS[0];
assign AISB = AutoAIS & RxAISL4M | UpAIS[1];
assign AISC = AutoAIS & RxAISL4M | UpAIS[2];
assign AISD = AutoAIS & RxAISL4M | UpAIS[3];

assign AlrmByteA = {UsPrtyErr[0],AISA,CritPwrAlm4M,5'b00000};
assign AlrmByteB = {UsPrtyErr[1],AISB,CritPwrAlm4M,5'b00000};
assign AlrmByteC = {UsPrtyErr[2],AISC,CritPwrAlm4M,5'b00000};
assign AlrmByteD = {UsPrtyErr[3],AISD,CritPwrAlm4M,5'b00000};

////////////////////////////////////////////////////////////////////////////////
// Output byte select signals and byte muxes
//
//  Interface C is different then the others -- see holding reg comment above
////////////////////////////////////////////////////////////////////////////////

assign RsrvBSel     = BitCnt[8:3] == 6'd1;
assign PrtyBSel     = BitCnt[8:3] == 6'd2;
assign MsgBSel      = BitCnt[8:3] == 6'd63;
assign AlrmBSel     = BitCnt[8:3] == 6'd0;

assign PCMSigBSel   = ~RsrvBSel & ~PrtyBSel & ~MsgBSel & ~AlrmBSel;
assign PCMBSel      = PCMSigBSel & BitCnt[3];
assign SigBSel      = PCMSigBSel & ~BitCnt[3];

assign MsgBCmdSel   = MsgBSel & FrmCnt[1:0] == 2'b00;
assign MsgBHAddSel  = MsgBSel & FrmCnt[1:0] == 2'b01;
assign MsgBLAddSel  = MsgBSel & FrmCnt[1:0] == 2'b10;
assign MsgBWrDSel   = MsgBSel & FrmCnt[1:0] == 2'b11;

assign PCMBSelNrmA  = PCMBSel & ~AISA;
assign PCMBSelAISA  = PCMBSel & AISA;
assign SigBSelNrmA  = SigBSel & ~AISA;
assign SigBSelAISA  = SigBSel & AISA;
assign MsgBCmdSelA  = MsgEnA & MsgBCmdSel;
assign MsgBHAddSelA = MsgEnA & MsgBHAddSel;
assign MsgBLAddSelA = MsgEnA & MsgBLAddSel;
assign MsgBWrDSelA  = MsgEnA & MsgBWrDSel;

assign PCMBSelNrmB  = PCMBSel & ~AISB;
assign PCMBSelAISB  = PCMBSel & AISB;
assign SigBSelNrmB  = SigBSel & ~AISB;
assign SigBSelAISB  = SigBSel & AISB;
assign MsgBCmdSelB  = MsgEnB & MsgBCmdSel;
assign MsgBHAddSelB = MsgEnB & MsgBHAddSel;
assign MsgBLAddSelB = MsgEnB & MsgBLAddSel;
assign MsgBWrDSelB  = MsgEnB & MsgBWrDSel;

assign PCMBSelNrmC  = PCMBSel & ~AISC;
assign PCMBSelAISC  = PCMBSel & AISC;
assign SigBSelNrmC  = SigBSel & ~AISC;
assign SigBSelAISC  = SigBSel & AISC;
assign MsgBCmdSelC  = MsgEnC & MsgBCmdSel;
assign MsgBHAddSelC = MsgEnC & MsgBHAddSel;
assign MsgBLAddSelC = MsgEnC & MsgBLAddSel;
assign MsgBWrDSelC  = MsgEnC & MsgBWrDSel;

assign PCMBSelNrmD  = PCMBSel & ~AISD;
assign PCMBSelAISD  = PCMBSel & AISD;
assign SigBSelNrmD  = SigBSel & ~AISD;
assign SigBSelAISD  = SigBSel & AISD;
assign MsgBCmdSelD  = MsgEnD & MsgBCmdSel;
assign MsgBHAddSelD = MsgEnD & MsgBHAddSel;
assign MsgBLAddSelD = MsgEnD & MsgBLAddSel;
assign MsgBWrDSelD  = MsgEnD & MsgBWrDSel;

assign ShftOByteA = {RsrvByte   & {8{RsrvBSel}}}
                    {PrtyByteA  & {8{PrtyBSel}}}
                    {PCMHldRegA & {8{PCMBSelNrmA}}}
                    {PCMAISByte & {8{PCMBSelAISA}}}
                    {SigHldRegA & {8{SigBSelNrmA}}}
                    {SigAISByte & {8{SigBSelAISA}}}
                    {MsgHldCmd  & {8{MsgBCmdSelA}}}
                    {MsgHldHAdd & {8{MsgBHAddSelA}}}
                    {MsgHldLAdd & {8{MsgBLAddSelA}}}
                    {MsgHldWrD  & {8{MsgBWrDSelA}}}
                    {AlrmByteA  & {8{AlrmBSel}}};

assign ShftOByteB = {RsrvByte   & {8{RsrvBSel}}}
                    {PrtyByteB  & {8{PrtyBSel}}}
                    {PCMHldRegB & {8{PCMBSelNrmB}}}
                    {PCMAISByte & {8{PCMBSelAISB}}}
                    {SigHldRegB & {8{SigBSelNrmB}}}
                    {SigAISByte & {8{SigBSelAISB}}}
                    {MsgHldCmd  & {8{MsgBCmdSelB}}}
                    {MsgHldHAdd & {8{MsgBHAddSelB}}}
                    {MsgHldLAdd & {8{MsgBLAddSelB}}}
                    {MsgHldWrD  & {8{MsgBWrDSelB}}}
                    {AlrmByteB  & {8{AlrmBSel}}};

assign ShftOByteC = {RsrvByte   & {8{RsrvBSel}}}
                    {PrtyByteC  & {8{PrtyBSel}}}
                    {DnPDOutB   & {8{PCMBSelNrmC}}}
                    {PCMAISByte & {8{PCMBSelAISC}}}
                    {SigHldRegC & {8{SigBSelNrmC}}}
                    {SigAISByte & {8{SigBSelAISC}}}
                    {MsgHldCmd  & {8{MsgBCmdSelC}}}
                    {MsgHldHAdd & {8{MsgBHAddSelC}}}
                    {MsgHldLAdd & {8{MsgBLAddSelC}}}
                    {MsgHldWrD  & {8{MsgBWrDSelC}}}
                    {AlrmByteC  & {8{AlrmBSel}}};

assign ShftOByteD = {RsrvByte   & {8{RsrvBSel}}}
                    {PrtyByteD  & {8{PrtyBSel}}}
                    {PCMHldRegD & {8{PCMBSelNrmD}}}
                    {PCMAISByte & {8{PCMBSelAISD}}}
                    {SigHldRegD & {8{SigBSelNrmD}}}
                    {SigAISByte & {8{SigBSelAISD}}}
                    {MsgHldCmd  & {8{MsgBCmdSelD}}}
                    {MsgHldHAdd & {8{MsgBHAddSelD}}}
                    {MsgHldLAdd & {8{MsgBLAddSelD}}}
                    {MsgHldWrD  & {8{MsgBWrDSelD}}}
                    {AlrmByteD  & {8{AlrmBSel}}};

////////////////////////////////////////////////////////////////////////////////
// Byte load signal for output shift registers
////////////////////////////////////////////////////////////////////////////////

assign LdShftRegs =  BitCnt[2:0] == 3'b111;

////////////////////////////////////////////////////////////////////////////////
// Output shift registers
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
   begin
   if (!RESET4M_)
```

```
        begin
        TDDAReg <= 8'h00;
        end
    else if (LdShftRegs)
        begin
        TDDAReg <= ShftOByteA;
        end
    else
        begin
        TDDAReg <= {TDDAReg[6:0],1'b0};
        end
end

assign oTDDA = TDDAReg[7];

always @ (posedge Clk4 or negedge RESET4M_)
begin
if (!RESET4M_)
    begin
    TDDBReg <= 8'h00;
    end
else if (LdShftRegs)
    begin
    TDDBReg <= ShftOByte8;
    end
else
    begin
    TDDBReg <= {TDDBReg[6:0],1'b0};
    end
end

assign oTDDB = TDDBReg[7];

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        TDDCReg <= 8'h00;
        end
    else if (LdShftRegs)
        begin
        TDDCReg <= ShftOByteC;
        end
    else
        begin
        TDDCReg <= {TDDCReg[6:0],1'b0};
        end
    end

assign oTDDC = TDDCReg[7];

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        TDDDReg <= 8'h00;
        end
    else if (LdShftRegs)
        begin
        TDDDReg <= ShftOByteD;
        end
    else
        begin
        TDDDReg <= {TDDDReg[6:0],1'b0};
        end
    end

assign oTDDD = TDDDReg[7];

endmodule // tdstint
```

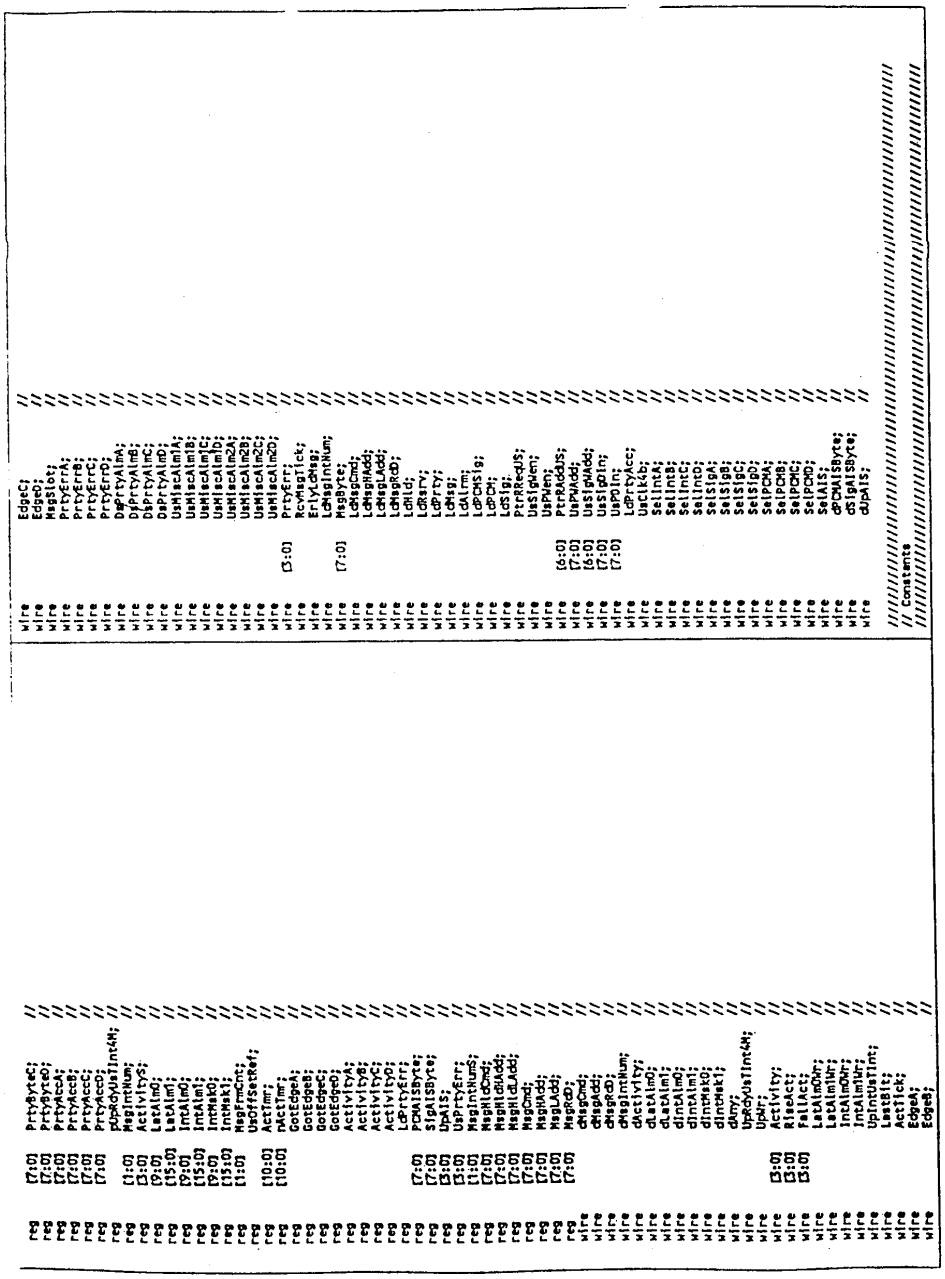
부록 C

```
//
// tptrrmctl.v :   Pointer RAM control
//
//
//
//
//
// functional desription:
//
//
//
// synthesis constraints
//
// sc   ClkList = {Clk4}
// sc   create_clock Clk4 -period 220 -waveform {0 110}
// sc   set_clock_skew -uncertainty 3 ClkList
// sc   set_fix_hold ClkList
// sc   set_false_path -from {RESET4M_}
// sc   set_dont_touch_network ClkList
// sc   set_dont_touch_network RESET4M_
// sc   set_load 32 * load_of(m5c/CMDF00/D) PtrClk4b

module tptrrmctl
    (
    RESET4M_,
    Clk4,
    PtrClk4b,
    PtrRAddDS,
    PtrRRenDS,
    PtrRAddUS,
    PtrRReqUS,
    PtrRAckUS,
    PtrRAdd,
    PtrRDOut,
    PtrRDOutDUS,
    PtrRCeb,
    PtrRWRb,
    UpReq4M,
    UpRdyPtrRmCtl4M,
    fUPA,
    fUPRNW,
    fUPD,
    UpRDPtrRmCtl
    );

input           RESET4M_;       //
input           Clk4;           //
output          PtrClk4b;       //
input   [6:0]   PtrRAddDS;      //
input           PtrRRenDS;      //
input   [6:0]   PtrRAddUS;      //
input           PtrRReqUS;      //
output          PtrRAckUS;      //
output  [6:0]   PtrRAdd;        //
input   [8:0]   PtrRDOut;       //
output  [8:0]   PtrRDOutDUS;    //
output          PtrRCeb;        //
output          PtrRWRb;        //
input           UpReq4M;        //
output          UpRdyPtrRmCtl4M;//
input   [10:1]  fUPA;           //
input           fUPRNW;         //
input   [15:0]  fUPD;           //
output  [15:0]  UpRDPtrRmCtl;   //

// local declarations

reg             dUpReq4M;       //
reg             UpPtrRmReqLt;   //
reg             DataEn;         //

wire            PtrRmHit;       //
wire            UpPtrRmReqSt;   //
wire            UpPtrRmReq;     //
wire            UpRdyPtrRmCtl4M;//
wire            PtrRAckUS;      //
wire            PtrRCeb;        //
wire            PtrRWRb;        //
wire    [6:0]   PtrRAdd;        //
wire    [8:0]   PtrRDOutDUS;    //
wire    [15:0]  UpRDPtrRmCtl;   //
wire            CtrlHit;        //
wire            CtrlRdy;        //
wire            UpPtrRmRdy;     //
wire            CtrlWr;         //
wire            PtrClk4b;       //

////////////////////////////////////////////////////////////////////////////////
// Clock inversion for pointer RAM
////////////////////////////////////////////////////////////////////////////////

assign PtrClk4b = ~Clk4;

////////////////////////////////////////////////////////////////////////////////
// Address decode
////////////////////////////////////////////////////////////////////////////////

assign PtrRmHit = UpReq4M & (fUPA[10:8] == 3'b001;
assign CtrlHit  = UpReq4M & (fUPA[10:1] == 10'b000_1000_000;

////////////////////////////////////////////////////////////////////////////////
// Cpu request for pointer RAM signal
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        dUpReq4M <= 1'b0;
    else
        dUpReq4M <= UpReq4M;

assign UpPtrRmReqSt = ~dUpReq4M & PtrRmHit;
assign CtrlRdy      = ~dUpReq4M & CtrlHit;

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        UpPtrRmReqLt <= 1'b0;
    else if (UpPtrRmRdy)
            UpPtrRmReqLt <= 1'b0;
        else if (UpPtrRmReqSt)
            UpPtrRmReqLt <= 1'b1;

assign UpPtrRmReq = UpPtrRmReqSt | UpPtrRmReqLt;

////////////////////////////////////////////////////////////////////////////////
// Arbitration between Cpu and Up-stream
// Up-stream has priority over the Cpu
////////////////////////////////////////////////////////////////////////////////

assign UpPtrRmRdy = ~PtrRRenDS & ~PtrRReqUS & UpPtrRmReq;

assign UpRdyPtrRmCtl4M = UpPtrRmRdy | CtrlRdy;
```

```
assign PtrRAckUS = ~PtrRRenDS & PtrRReqUS;

////////////////////////////////////////////////////////////////////////////////
// Pointer RAM Add mux and control signals
////////////////////////////////////////////////////////////////////////////////

assign PtrRAdd =    {PtrRAddS & {7{PtrRRenDS}}} |
                    {PtrRAddUS & {7{PtrRAckUS}}} |
                    {lUPA[7:1] & {7{UpPtrRmRdy}}};

assign PtrREb = ~(PtrRRenDS | PtrRAckUS | UpPtrRmRdy);

assign PtrRWrb = ~lUPRNW & UpPtrRmRdy;

////////////////////////////////////////////////////////////////////////////////
// Pointer RAM data disable function
////////////////////////////////////////////////////////////////////////////////

assign PtrRDOutDUS =   {PtrRDOut & {9{DataEn}}};

////////////////////////////////////////////////////////////////////////////////
// Control register
////////////////////////////////////////////////////////////////////////////////

assign CtrlWr = ~lUPRNW & CtrlRdy;

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        DataEn <= 1'b0;
    else if (CtrlWr)
        DataEn <= lUPD[0];

////////////////////////////////////////////////////////////////////////////////
// Cpu read data mux
////////////////////////////////////////////////////////////////////////////////

assign UpRDPtrRmCtl =   {{7'd0,PtrRDOut} & {16{PtrRmRt}}} |
                        {{15'd0,DataEn} & {16{Ctrlrlkrt}}};

endmodule // tptrrmctl
```

부록 D

```
//
// tuatint.v 1  Up-stream telephony interface
//
//
//
//
// functional description:
//
//
// synthesis constraints
//
// sc  clklist = {clk4 UsClk4b}
// sc  create_clock clk4 -period 220 -waveform {0 110}
// sc  set_clock_skew -uncertainty 3 clklist
// sc  set_fix_hold clklist
// sc  set_false_path -from {RESET4M_}
// sc  set_dont_touch_network clklist
// sc  set_dont_touch_network RESET4M
// sc  set_load 32 * [get_nets -of_objects [get_pins -leaf -filter {DIRECTION==OUT} -of_objects UsClk4b
// sc  set_input_delay 1 {UsFrmCnt}
module tuatint
    (
    RESET4M_,
    Clk4,
    UsClk4b,
    UsCntrRst,
    UsFrmRef,
    UsFrmCnt,
    DnMsgFlc,
    BitOffSetChg,
    UsOffSetRef,
    UsPDin,
    UsPUAdd,
    UsPWen,
    UsSigDin,
    UsSigUAdd,
    UsSigWen,
    PtrRAddUS,
    PtrRDOutDUS,
    PtrRReqUS,
    PtrRAckUS,
    UsSigRdBlk4M,
    ITUDA,
    ITUDB,
    ITUDC,
    ITUDD,
    UsPrtyErr,
    UpReq4M,
    UpRdyUsInt4M,
    !UPA,
    !UPRN,
    !UPD,
    UpRDUsTInt,
    UpIntUsTInt
    );


input       RESET4M_;
input       Clk4;
input       UsCntrRst;  //
input [2:0] UsFrmCnt;   //
input [1:0] unwyrij
input       BitOffSetChg;
input [8:0] PtrRDOutDUS;
input       PtrRAckUS;
input       ITUDA;                  //Serial input data for telephony interface A
input       ITUDC;                  //Serial input data for telephony interface B
input       ITUDB;                  //Serial input data for telephony interface C
input       ITUDD;                  //Serial input data for telephony interface D
input       UpReq4M;
input [10:1] !UPA;
input       !UPRN;
input [15:0] !UPD;

output      UsClk4b;                //
output [7:0] UsPDin;                //Up-stream PCM RAM Data-In
output [7:0] UsPUAdd;               //Up-stream PCM RAM Write Address-In
output      UsPWen;                 //Up-stream PCM RAM Write Enable
output      UsFrmRef;               //Up-stream frame reference pulse
output      UsOffSetRef;            //
output [7:0] UsSigDin;              //Up-stream Signalling RAM Data-In
output [6:0] UsSigUAdd;             //Up-stream Signalling RAM Write Address-In
output      UsSigWen;               //Up-stream Signalling RAM Write Enable
output [6:0] PtrRAddUS;             //
output      PtrReqUS;               //
output [3:0] UsPrtyErr;             //
output      UsSigRdBlk4M;           //
output      UpRdyUsInt4M;           //
output [15:0] UpRDUsTInt;           //
output      UpIntUsTInt;            //

// local declarations

reg  [8:0] rdItCnt;                 //
reg  [8:0] BitCnt;                  //
reg  [2:0] FrmCnt;                  //
reg        UsFrmRef;                //
reg        TUDA;                    //
reg        TUDB;                    //
reg        TUDC;                    //
reg        TUDD;                    //
reg  [7:0] TUDAReg;                 //
reg  [7:0] TUDBReg;                 //
reg  [7:0] TUDCReg;                 //
reg  [7:0] TUDDReg;                 //
reg  [7:0] PCMHldRegA;              //
reg  [7:0] PCMHldRegB;              //
reg  [7:0] PCMHldRegC;              //
reg  [7:0] SigHldRegB;              //
reg  [7:0] SigHldRegC;              //
reg  [7:0] SigHldRegD;              //
reg  [7:0] PrtyHldRegA;             //
reg  [7:0] PrtyHldRegB;             //
reg  [7:0] PrtyHldRegC;             //
reg  [7:0] PrtyHldRegD;             //
reg  [6:0] UsPUAddLwr;              //
reg  [1:0] IntAdd;                  //
reg  [1:0] PtrRAddLwr;              //
reg        PtrRReqUSa;              //
reg        PtrRReqUSb;              //
reg        UsSigRdBlk4M;            //
reg        SigEn;                   //
reg        PCMEn;                   //
reg        UsWen;                   //
reg  [7:0] PrtyByteA;               //
reg  [7:0] PrtyByteB;               //
```



```
parameter cOffSetRef = 9'd485;

////////////////////////////////////////////////////////////////////////////////
// Clock inversion for up-stream signalling and PCM RAMs
////////////////////////////////////////////////////////////////////////////////

assign UsClk4b = ~Clk4;

////////////////////////////////////////////////////////////////////////////////
// Processor address decodes
////////////////////////////////////////////////////////////////////////////////

assign dAny          = UpReq4M & (iUPA[10:6] == 5'b000_01);
assign dActivity     = dAny & (iUPA[5:1] == 5'b00_000);
assign dLatAlm0      = dAny & (iUPA[5:1] == 5'b00_001);
assign dLatAlm1      = dAny & (iUPA[5:1] == 5'b00_010);
assign dIntAlm0      = dAny & (iUPA[5:1] == 5'b00_011);
assign dIntAlm1      = dAny & (iUPA[5:1] == 5'b00_100);
assign dIntMsk0      = dAny & (iUPA[5:1] == 5'b00_101);
assign dIntMsk1      = dAny & (iUPA[5:1] == 5'b00_110);
assign dMsgIntNum    = dAny & (iUPA[5:1] == 5'b01_000);
assign dMsgCmd       = dAny & (iUPA[5:1] == 5'b01_001);
assign dMsgAdd       = dAny & (iUPA[5:1] == 5'b01_010);
assign dMsgRdD       = dAny & (iUPA[5:1] == 5'b01_011);
assign dPCMAISByte   = dAny & (iUPA[5:1] == 5'b01_100);
assign dSigAISByte   = dAny & (iUPA[5:1] == 5'b01_101);
assign dUpAIS        = dAny & (iUPA[5:1] == 5'b01_110);

////////////////////////////////////////////////////////////////////////////////
// Porcessor interface control logic, storage regs, and read mux
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        pUpRdyUsTint4M <= 1'b0;
        end
    else
        begin
        pUpRdyUsTint4M <= UpReq4M;
        end
    end

assign UpRdyUsTint4M = dAny & UpReq4M & ~pUpRdyUsTint4M;
assign UpWr      = UpRdyUsTint4M & ~iUPRNW;

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        PCMAISByte <= 8'h7f;
        end
    else if (UpWr & dPCMAISByte)
        begin
        PCMAISByte <= iUPD[7:0];
        end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        SigAISByte <= 8'h02;
        end
    else if (UpWr & dSigAISByte)
        begin
        SigAISByte <= iUPD[7:0];
        end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        UpAIS <= 4'hf;
        end
    else if (UpWr & dUpAIS)
        begin
        UpAIS <= iUPD[3:0];
        end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        MsgIntNum <= 2'b00;
        end
    else if (UpWr & dMsgIntNum)
        begin
        MsgIntNum <= iUPD[1:0];
        end
    end

assign LatAlm0Wr = UpWr & dLatAlm0;

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        LatAlm0 <= 10'd0;
        end
    else
        begin
        LatAlm0[0] <= LatAlm0[0] ? ~(~BitOffSetChg& LatAlm0Wr & iUPD[0]) : BitOffSetC
        LatAlm0[1] <= LatAlm0[1] ? ~(~RcvMsgTick    & LatAlm0Wr & iUPD[1]) : RcvMsgTi
        LatAlm0[2] <= LatAlm0[2] ? ~(~UsMiscAlm1A   & LatAlm0Wr & iUPD[2]) : UsMiscAl
        LatAlm0[3] <= LatAlm0[3] ? ~(~UsMiscAlm1B   & LatAlm0Wr & iUPD[3]) : UsMiscAl
        LatAlm0[4] <= LatAlm0[4] ? ~(~UsMiscAlm1C   & LatAlm0Wr & iUPD[4]) : UsMiscAl
        LatAlm0[5] <= LatAlm0[5] ? ~(~UsMiscAlm1D   & LatAlm0Wr & iUPD[5]) : UsMiscA
        LatAlm0[6] <= LatAlm0[6] ? ~(~UsMiscAlm2A   & LatAlm0Wr & iUPD[6]) : UsMiscA
        LatAlm0[7] <= LatAlm0[7] ? ~(~UsMiscAlm2B   & LatAlm0Wr & iUPD[7]) : UsMiscAl
        LatAlm0[8] <= LatAlm0[8] ? ~(~UsMiscAlm2C   & LatAlm0Wr & iUPD[8]) : UsMiscAl
        LatAlm0[9] <= LatAlm0[9] ? ~(~UsMiscAlm2D   & LatAlm0Wr & iUPD[9]) : UsMiscAl
        end
    end

assign Activity = {ActivityD,ActivityC,ActivityB,ActivityA};

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        ActivityS <= 4'h0;
        end
    else
        begin
        ActivityS <= Activity;
        end
    end
```


등록특허 10-0507023

```
assign RiseAct = Activity & ~Activity5;
assign FallAct = ~Activity & Activity5;

assign LatAlm1Wr = UpWr & dLatAlm1;

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        LatAlm1 <= 16'd0;
        end
    else
        begin
        LatAlm1[0] <= LatAlm1[0] ? ~(~FallAct[0]    & LatAlm1Wr & iUPD[0]) : FallAct[
        LatAlm1[1] <= LatAlm1[1] ? ~(~RiseAct[0]    & LatAlm1Wr & iUPD[1]) : RiseAct[
        LatAlm1[2] <= LatAlm1[2] ? ~(~DsPrtyAlmA    & LatAlm1Wr & iUPD[2]) : DsPrtyAl
        LatAlm1[3] <= LatAlm1[3] ? ~(~PrtyErr[0]    & LatAlm1Wr & iUPD[3]) : PrtyErr[
        LatAlm1[4] <= LatAlm1[4] ? ~(~FallAct[1]    & LatAlm1Wr & iUPD[4]) : FallAct[
        LatAlm1[5] <= LatAlm1[5] ? ~(~RiseAct[1]    & LatAlm1Wr & iUPD[5]) : RiseAct[
        LatAlm1[6] <= LatAlm1[6] ? ~(~DsPrtyAlmB    & LatAlm1Wr & iUPD[6]) : DsPrtyAl
        LatAlm1[7] <= LatAlm1[7] ? ~(~PrtyErr[1]    & LatAlm1Wr & iUPD[7]) : PrtyErr[
        LatAlm1[8] <= LatAlm1[8] ? ~(~FallAct[2]    & LatAlm1Wr & iUPD[8]) : FallAct[
        LatAlm1[9] <= LatAlm1[9] ? ~(~RiseAct[2]    & LatAlm1Wr & iUPD[9]) : RiseAct[
        LatAlm1[10] <= LatAlm1[10] ? ~(~DsPrtyAlmC  & LatAlm1Wr & iUPD[10]) : DsPrtyA
        LatAlm1[11] <= LatAlm1[11] ? ~(~PrtyErr[2]  & LatAlm1Wr & iUPD[11]) : PrtyErr
        LatAlm1[12] <= LatAlm1[12] ? ~(~FallAct[3]  & LatAlm1Wr & iUPD[12]) : FallAct
        LatAlm1[13] <= LatAlm1[13] ? ~(~RiseAct[3]  & LatAlm1Wr & iUPD[13]) : RiseAct
        LatAlm1[14] <= LatAlm1[14] ? ~(~DsPrtyAlmD  & LatAlm1Wr & iUPD[14]) : DsPrtyA
        LatAlm1[15] <= LatAlm1[15] ? ~(~PrtyErr[3]  & LatAlm1Wr & iUPD[15]) : PrtyErr
        end
    end

assign IntAlm0Wr = UpWr & dIntAlm0;

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        IntAlm0 <= 10'd0;
        end
    else
        begin
        IntAlm0[0] <= IntAlm0[0] ? ~(IntAlm0Wr & iUPD[0]) : BitOffSetChg;
        IntAlm0[1] <= IntAlm0[1] ? ~(IntAlm0Wr & iUPD[1]) : RcvMsgTick;
        IntAlm0[2] <= IntAlm0[2] ? ~(IntAlm0Wr & iUPD[2]) : UsMiscAlm1A;
        IntAlm0[3] <= IntAlm0[3] ? ~(IntAlm0Wr & iUPD[3]) : UsMiscAlm1B;
        IntAlm0[4] <= IntAlm0[4] ? ~(IntAlm0Wr & iUPD[4]) : UsMiscAlm1C;
        IntAlm0[5] <= IntAlm0[5] ? ~(IntAlm0Wr & iUPD[5]) : UsMiscAlm1D;
        IntAlm0[6] <= IntAlm0[6] ? ~(IntAlm0Wr & iUPD[6]) : UsMiscAlm2A;
        IntAlm0[7] <= IntAlm0[7] ? ~(IntAlm0Wr & iUPD[7]) : UsMiscAlm2B;
        IntAlm0[8] <= IntAlm0[8] ? ~(IntAlm0Wr & iUPD[8]) : UsMiscAlm2C;
        IntAlm0[9] <= IntAlm0[9] ? ~(IntAlm0Wr & iUPD[9]) : UsMiscAlm2D;
        end
    end
assign IntAlm1Wr = UpWr & dIntAlm1;

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        IntAlm1 <= 16'd0;
        end
    else
        begin
        IntAlm1[0] <= IntAlm1[0] ? ~(IntAlm1Wr & iUPD[0]) : FallAct[0];
        IntAlm1[1] <= IntAlm1[1] ? ~(IntAlm1Wr & iUPD[1]) : RiseAct[0];
        IntAlm1[2] <= IntAlm1[2] ? ~(IntAlm1Wr & iUPD[2]) : DsPrtyAlmA;
        IntAlm1[3] <= IntAlm1[3] ? ~(IntAlm1Wr & iUPD[3]) : PrtyErr[0];
        IntAlm1[4] <= IntAlm1[4] ? ~(IntAlm1Wr & iUPD[4]) : FallAct[1];
        IntAlm1[5] <= IntAlm1[5] ? ~(IntAlm1Wr & iUPD[5]) : RiseAct[1];
        IntAlm1[6] <= IntAlm1[6] ? ~(IntAlm1Wr & iUPD[6]) : DsPrtyAlmB;
        IntAlm1[7] <= IntAlm1[7] ? ~(IntAlm1Wr & iUPD[7]) : PrtyErr[1];
        IntAlm1[8] <= IntAlm1[8] ? ~(IntAlm1Wr & iUPD[8]) : FallAct[2];
        IntAlm1[9] <= IntAlm1[9] ? ~(IntAlm1Wr & iUPD[9]) : RiseAct[2];
        IntAlm1[10] <= IntAlm1[10] ? ~(IntAlm1Wr & iUPD[10]) : DsPrtyAlmC;
        IntAlm1[11] <= IntAlm1[11] ? ~(IntAlm1Wr & iUPD[11]) : PrtyErr[2];
        IntAlm1[12] <= IntAlm1[12] ? ~(IntAlm1Wr & iUPD[12]) : FallAct[3];
        IntAlm1[13] <= IntAlm1[13] ? ~(IntAlm1Wr & iUPD[13]) : RiseAct[3];
        IntAlm1[14] <= IntAlm1[14] ? ~(IntAlm1Wr & iUPD[14]) : DsPrtyAlmD;
        IntAlm1[15] <= IntAlm1[15] ? ~(IntAlm1Wr & iUPD[15]) : PrtyErr[3];
        end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        IntMsk0 <= 10'd0;
        end
    else if (UpWr & dIntMsk0)
        begin
        IntMsk0 <= iUPD[9:0];
        end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        IntMsk1 <= 16'd0;
        end
    else if (UpWr & dIntMsk1)
        begin
        IntMsk1 <= iUPD[15:0];
        end
    end

assign UpRDUsTInt = ((8'h00,MsgCmd)     & (16{dMsgCmd}))
                    ((MsgHAdd,MsgLAdd)  & (16{dMsgAdd}))
                    ((8'h00,MsgRcD)     & (16{dMsgRcD}))
                    ((14'd0,MsgIntNum)  & (16{dMsgIntNum}))
                    ((8'h00,PCMAISByte) & (16{dPCMAISByte}))
                    ((8'h00,SigAISByte) & (16{dSigAISByte}))
                    ((12'd0,UpAIS)      & (16{dUpAIS}))
                    ((12'd0,Activity)   & (16{dActivity}))
                    ((6'd0,LatAlm0)     & (16{dLatAlm0}))
                    (LatAlm1            & (16{dLatAlm1}))
                    ((6'd0,IntAlm0)     & (16{dIntAlm0}))
                    (IntAlm1            & (16{dIntAlm1}))
                    ((6'd0,IntMsk0)     & (16{dIntMsk0}))
                    (IntMsk1            & (16{dIntMsk1}));

assign UpIntUsTInt = (| (IntAlm0 & IntMsk0)) | (| (IntAlm1 & IntMsk1));

////////////////////////////////////////////////////////////////////////////
// Master bit counter, reference frame pulse and frame count register
////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
```

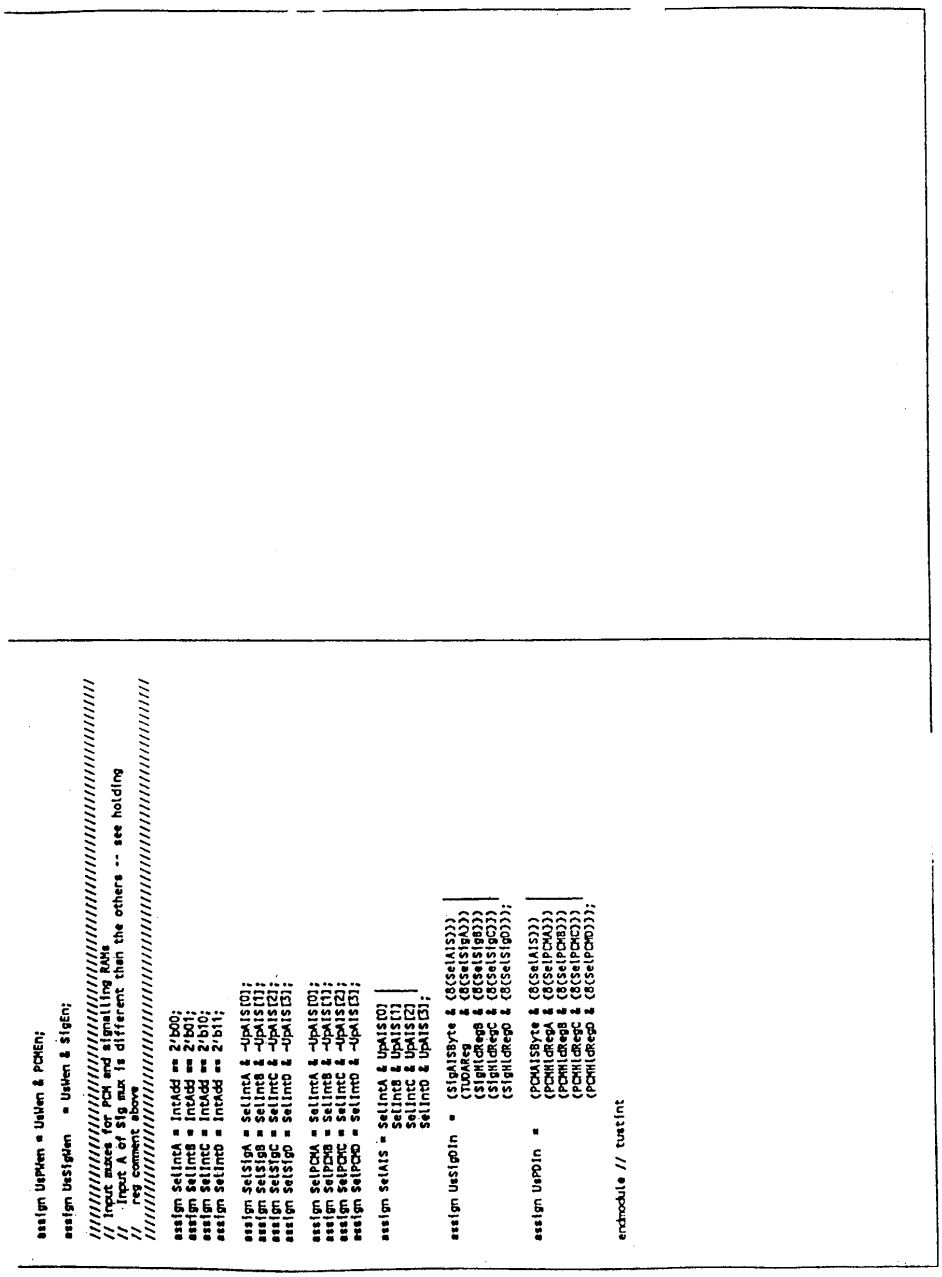
```
        begin
          BitCnt <= 9'h00;
          UaFrmRef <= 1'b0;
        end
      else
        begin
          BitCnt <= nBitCnt;
          UaFrmRef <= (BitCnt == 9'b11101_0000);
        end
    end

  always @ (BitCnt or UaCntrRst)
    begin
      if (UaCntrRst)
        nBitCnt = 9'h00;
      else
        nBitCnt = BitCnt + 1'b1;
    end

  assign LastBit = (BitCnt == 9'd511);

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
      if (!RESET4M_)
        FrmCnt <= 3'b000;
      else if (UaCntrRst | LastBit)
        FrmCnt <= UaFrmCnt;
    end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
      if (!RESET4M_)
        MsgFrmCnt <= 2'b00;
      else if (LastBit)
        MsgFrmCnt <= DrMsgFC;
    end

  ////////////////////////////////////////////////////////////////////
  // Reference pulse used by down-stream module to capture bit offset for TUA
  ////////////////////////////////////////////////////////////////////

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
      if (!RESET4M_)
        UaOffSetRef <= 1'b0;
      else
        UaOffSetRef <= BitCnt == cOffSetRef;
    end

  ////////////////////////////////////////////////////////////////////
  // Input shift registers with data clocked in on the falling edge of the 4MHz
  ////////////////////////////////////////////////////////////////////

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
      if (!RESET4M_)
        begin
          TUDA <= 1'b0;
          TUDB <= 1'b0;
          TUDC <= 1'b0;
          TUDD <= 1'b0;
        end
      else
        begin
          TUDA <= !TUDA;
          TUDB <= !TUDB;
          TUDC <= !TUDC;
          TUDD <= !TUDD;
        end
    end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
      if (!RESET4M_)
        begin
          TUDARegE <= 8'h00;
        end
      else
        begin
          TUDARegE <= {TUDARegE[6:0],TUDA};
        end
    end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
      if (!RESET4M_)
        begin
          TUDBRegE <= 8'h00;
        end
      else
        begin
          TUDBRegE <= {TUDBRegE[6:0],TUDB};
        end
    end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
      if (!RESET4M_)
        begin
          TUDCRegE <= 8'h00;
        end
      else
        begin
          TUDCRegE <= {TUDCRegE[6:0],TUDC};
        end
    end

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
      if (!RESET4M_)
        begin
          TUDDRegE <= 8'h00;
        end
      else
        begin
          TUDDRegE <= {TUDDRegE[6:0],TUDD};
        end
    end

  ////////////////////////////////////////////////////////////////////
  // Activity detectors -- at least one edge must occur every four frames
  ////////////////////////////////////////////////////////////////////

  always @ (posedge Clk4 or negedge RESET4M_)
    begin
      if (!RESET4M_)
        begin
          ActTmr <= 11'd0;
        end
      else
        begin
          ActTmr <= nActTmr;
```

```
        end
    end

always @ (ActTmr)
    begin
    nActTmr = ActTmr + 1'b1;
    end

assign ActTick = ~nActTmr[10] & ActTmr[10];

assign EdgeA = TUDAReg[1] != TUDAReg[0];
assign EdgeB = TUDBReg[1] != TUDBReg[0];
assign EdgeC = TUDCReg[1] != TUDCReg[0];
assign EdgeD = TUDDReg[1] != TUDDReg[0];

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        GotEdgeA <= 1'b0;
    else if (EdgeA)
            GotEdgeA <= 1'b1;
        else if (ActTick)
                GotEdgeA <= 1'b0;

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        GotEdgeB <= 1'b0;
    else if (EdgeB)
            GotEdgeB <= 1'b1;
        else if (ActTick)
                GotEdgeB <= 1'b0;

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        GotEdgeC <= 1'b0;
    else if (EdgeC)
            GotEdgeC <= 1'b1;
        else if (ActTick)
                GotEdgeC <= 1'b0;

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        GotEdgeD <= 1'b0;
    else if (EdgeD)
            GotEdgeD <= 1'b1;
        else if (ActTick)
                GotEdgeD <= 1'b0;

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        ActivityA <= 1'b0;
        ActivityB <= 1'b0;
        ActivityC <= 1'b0;
        ActivityD <= 1'b0;
        end
    else if (ActTick)
            begin
            ActivityA <= GotEdgeA;
            ActivityB <= GotEdgeB;
            ActivityC <= GotEdgeC;
            ActivityD <= GotEdgeD;
            end

////////////////////////////////////////////////////////////////////////
// Load signals for holding registers
////////////////////////////////////////////////////////////////////////

assign  LdHld       = BitCnt[2:0] == 3'b111;
assign  LdRsrv      = LdHld & BitCnt[8:3] == 6'd63;
assign  LdPrty      = LdHld & BitCnt[8:3] == 6'd0;
assign  MsgSlot     = BitCnt[8:3] == 6'd61;
assign  LdAlrm      = LdHld & BitCnt[8:3] == 6'd62;
assign  LdPCMSig    = ~LdRsrv & ~LdPrty & ~LdMsg & ~LdAlrm;
assign  LdPCM       = LdHld & LdPCMSig & BitCnt[3];
assign  LdSig       = LdHld & LdPCMSig & ~BitCnt[3];

////////////////////////////////////////////////////////////////////////
// Holding registers for PCM and signalling
//
//  Sig holding reg for interface A is not needed because the Sig buffer
//  write occurs just as the shift byte arrives thus no holding reg is
//  required.
////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        PCMHldRegA <= 8'h00;
        PCMHldRegB <= 8'h00;
        PCMHldRegC <= 8'h00;
        PCMHldRegD <= 8'h00;
        end
    else if (LdPCM)
            begin
            PCMHldRegA <= TUDAReg;
            PCMHldRegB <= TUDBReg;
            PCMHldRegC <= TUDCReg;
            PCMHldRegD <= TUDDReg;
            end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        SigHldRegB <= 8'h00;
        SigHldRegC <= 8'h00;
        SigHldRegD <= 8'h00;
        end
    else if (LdSig)
            begin
            SigHldRegB <= TUDBReg;
            SigHldRegC <= TUDCReg;
            SigHldRegD <= TUDDReg;
            end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        PrtyHldRegA <= 8'h00;
        PrtyHldRegB <= 8'h00;
        PrtyHldRegC <= 8'h00;
        PrtyHldRegD <= 8'h00;
        end
    else if (LdPrty)
            begin
            PrtyHldRegA <= TUDAReg;
            PrtyHldRegB <= TUDBReg;
            PrtyHldRegC <= TUDCReg;
```

```
            PrtyHldRegD <= TUDDReg;
            end
    end

////////////////////////////////////////////////////////////////////////////////
// Parity generation and storage
////////////////////////////////////////////////////////////////////////////////

assign LdPrtyAcc = LdRsrv & LdHld;

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        PrtyByteA <= 8'h00;
        PrtyByteB <= 8'h00;
        PrtyByteC <= 8'h00;
        PrtyByteD <= 8'h00;
        PrtyAccA <= 8'h00;
        PrtyAccB <= 8'h00;
        PrtyAccC <= 8'h00;
        PrtyAccD <= 8'h00;
        end
    else if (LdPrtyAcc)
            begin
            PrtyByteA <= PrtyAccA;
            PrtyByteB <= PrtyAccB;
            PrtyByteC <= PrtyAccC;
            PrtyByteD <= PrtyAccD;
            PrtyAccA <= ~TUDAReg;
            PrtyAccB <= ~TUDBReg;
            PrtyAccC <= ~TUDCReg;
            PrtyAccD <= ~TUDDReg;
            end
        else if (LdHld)
            begin
            PrtyAccA <= PrtyAccA ^ TUDAReg;
            PrtyAccB <= PrtyAccB ^ TUDBReg;
            PrtyAccC <= PrtyAccC ^ TUDCReg;
            PrtyAccD <= PrtyAccD ^ TUDDReg;
            end
    end

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        LdPrtyErr <= 1'b0;
        end
    else
        begin
        LdPrtyErr <= LdPrty;
        end

assign PrtyErrA = LdPrtyErr & PrtyByteA != PrtyHldRegA;
assign PrtyErrB = LdPrtyErr & PrtyByteB != PrtyHldRegB;
assign PrtyErrC = LdPrtyErr & PrtyByteC != PrtyHldRegC;
assign PrtyErrD = LdPrtyErr & PrtyByteD != PrtyHldRegD;

assign PrtyErr = {PrtyErrD, PrtyErrC, PrtyErrB, PrtyErrA};

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        UsPrtyErr <= 4'h0;
        end
    else if (LdPrtyErr)
        begin
        UsPrtyErr <= PrtyErr;
        end

////////////////////////////////////////////////////////////////////////////////
// Message control and storage
////////////////////////////////////////////////////////////////////////////////

assign RcvMsgTick = LdPrty & MsgFrmCnt == 2'b00;

assign ErlyLdMsg    = MsgSlot & BitCnt[2:0] == 3'b110;

assign LdMsgIntNum  = ErlyLdMsg & MsgFrmCnt[1:0] == 2'b00;

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        MsgIntNumS <= 2'b00;
        end
    else if (LdMsgIntNum)
        begin
        MsgIntNumS <= MsgIntNum;
        end

assign MsgByte  =   (TUDAReg & {8{MsgIntNumS == 2'b00}}) |
                    (TUDBReg & {8{MsgIntNumS == 2'b01}}) |
                    (TUDCReg & {8{MsgIntNumS == 2'b10}}) |
                    (TUDDReg & {8{MsgIntNumS == 2'b11}});

assign LdMsg        = MsgSlot & LdHld;

assign LdMsgCmd     = LdMsg & MsgFrmCnt == 2'b00;
assign LdMsgHAdd    = LdMsg & MsgFrmCnt == 2'b01;
assign LdMsgLAdd    = LdMsg & MsgFrmCnt == 2'b10;
assign LdMsgRdD     = LdMsg & MsgFrmCnt == 2'b11;

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        MsgHldCmd <= 8'h00;
        MsgHldHAdd <= 8'h00;
        MsgHldLAdd <= 8'h00;
        end
    else
        begin
        MsgHldCmd <=  (LdMsgCmd) ? MsgByte : MsgHldCmd;
        MsgHldHAdd <=  (LdMsgHAdd) ? MsgByte : MsgHldHAdd;
        MsgHldLAdd <=  (LdMsgLAdd) ? MsgByte : MsgHldLAdd;
        end

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        MsgCmd <= 8'h00;
        MsgHAdd <= 8'h00;
        MsgLAdd <= 8'h00;
        MsgRdD <= 8'h00;
        end
    else if (LdMsgRdD)
            begin
            MsgCmd <= MsgHldCmd;
            MsgHAdd <= MsgHldHAdd;
            MsgLAdd <= MsgHldLAdd;
            MsgRdD <= MsgByte;
            end
```


등록특허 10-0507023

```
////////////////////////////////////////////////////////////////////////////////
// Down-stream parity alarm
////////////////////////////////////////////////////////////////////////////////

assign DsPrtyAlmA = LdAlrm & TUDAReg[7];
assign DsPrtyAlmB = LdAlrm & TUDBReg[7];
assign DsPrtyAlmC = LdAlrm & TUDCReg[7];
assign DsPrtyAlmD = LdAlrm & TUDDReg[7];

assign UsMiscAlm1A = LdAlrm & TUDAReg[6];
assign UsMiscAlm1B = LdAlrm & TUDBReg[6];
assign UsMiscAlm1C = LdAlrm & TUDCReg[6];
assign UsMiscAlm1D = LdAlrm & TUDDReg[6];

assign UsMiscAlm2A = LdAlrm & TUDAReg[5];
assign UsMiscAlm2B = LdAlrm & TUDBReg[5];
assign UsMiscAlm2C = LdAlrm & TUDCReg[5];
assign UsMiscAlm2D = LdAlrm & TUDDReg[5];

////////////////////////////////////////////////////////////////////////////////
// Decode of bit counter to get misc control addresses and signals
////////////////////////////////////////////////////////////////////////////////

always @ (BitCnt)
    begin
    if (BitCnt[8:4] == 5'd31 | BitCnt[8:4] == 5'd0)
        begin
        PtrRAddLwr = 2'b00;
        PtrRReqUSa = 1'b0;
        end
    else
        case (BitCnt[3:0])
            4'b0101 : begin
                        PtrRAddLwr = 2'b00;
                        PtrRReqUSa = 1'b1;
                      end
            4'b0110 : begin
                        PtrRAddLwr = 2'b00;
                        PtrRReqUSa = 1'b0;
                      end
            4'b0111 : begin
                        PtrRAddLwr = 2'b01;
                        PtrRReqUSa = 1'b1;
                      end
            4'b1000 : begin
                        PtrRAddLwr = 2'b01;
                        PtrRReqUSa = 1'b0;
                      end
            4'b1010 : begin
                        PtrRAddLwr = 2'b10;
                        PtrRReqUSa = 1'b1;
                      end
            4'b1011 : begin
                        PtrRAddLwr = 2'b10;
                        PtrRReqUSa = 1'b0;
                      end
            4'b1100 : begin
                        PtrRAddLwr = 2'b11;
                        PtrRReqUSa = 1'b1;
                      end
            4'b1101 : begin
                        PtrRAddLwr = 2'b11;
                        PtrRReqUSa = 1'b0;
                      end
            default : begin
                        PtrRAddLwr = 2'b00;
                        PtrRReqUSa = 1'b0;
                      end
        endcase
    end

////////////////////////////////////////////////////////////////////////////////
// Generate pointer RAM signals using synchronous delays
// Also generate read blocking signal for other side of signalling RAM
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        PtrRReqUSb <= 1'b0;
        UsSigRdBlk4M <= 1'b0;
        end
    else
        begin
        PtrRReqUSb <= ~PtrRAckUS & PtrRReqUSa;
        UsSigRdBlk4M <= PtrRReqUSa;
        end

assign PtrRAddUS = {PtrRAddLwr,BitCnt[8:4]};

assign PtrRReqUS = PtrRReqUSa | PtrRReqUSb;

////////////////////////////////////////////////////////////////////////////////
// Stored info from pointer RAM
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk4 or negedge RESET4M_)
    begin
    if (!RESET4M_)
        begin
        UsPWAddLwr <= 7'h00;
        SigEn <= 1'b0;
        PCMEn <= 1'b0;
        end
    else if (PtrRAckUS)
            begin
            UsPWAddLwr <= PtrRDOutDUS[8:2];
            SigEn <= PtrRDOutDUS[1];
            PCMEn <= PtrRDOutDUS[0];
            end
    end

////////////////////////////////////////////////////////////////////////////////
// Generate PCM and signalling RAM signals using synchronous delays
////////////////////////////////////////////////////////////////////////////////

assign UsPWAdd = {FrmCnt[0],UsPWAddLwr};


always @ (posedge Clk4 or negedge RESET4M_)
    if (!RESET4M_)
        begin
        IntAdd <= 2'b00;
        UsWen <= 1'b0;
        end
    else
        begin
        IntAdd <= PtrRAddLwr;
        UsWen <= UsSigRdBlk4M;
        end

assign UsSigWAdd = UsPWAddLwr;
```



```
assign UsPCen = UsVen & PCMEn;
assign UsSigGen   = UsVen & SigEn;

////////////////////////////////////////////////////////////////////////////////
// Input muxes for PCM and signalling RAMs
//   Input A of Sig mux is different than the others -- see holding
//   reg comment above
////////////////////////////////////////////////////////////////////////////////

assign SelIntA = IntAdd == 2'b00;
assign SelIntB = IntAdd == 2'b01;
assign SelIntC = IntAdd == 2'b10;
assign SelIntD = IntAdd == 2'b11;

assign SelSigA = SelIntA & ~UpAIS[0];
assign SelSigB = SelIntB & ~UpAIS[1];
assign SelSigC = SelIntC & ~UpAIS[2];
assign SelSigD = SelIntD & ~UpAIS[3];

assign SelPCMA = SelIntA & ~UpAIS[0];
assign SelPCMB = SelIntB & ~UpAIS[1];
assign SelPCMC = SelIntC & ~UpAIS[2];
assign SelPCMD = SelIntD & ~UpAIS[3];

assign SetAIS = SelIntA & UpAIS[0] |
                SelIntB & UpAIS[1] |
                SelIntC & UpAIS[2] |
                SelIntD & UpAIS[3];

assign UsSigDin  =  {SigAISByte & {8{SelAIS}}} |
                    {TUDARcg    & {8{SelSigA}}} |
                    {SigHldRegB & {8{SelSigB}}} |
                    {SigHldRegC & {8{SelSigC}}} |
                    {SigHldRegD & {8{SelSigD}}};

assign UsPDin    =  {PCMAISByte & {8{SelAIS}}} |
                    {PCMHldRegA & {8{SelPCMA}}} |
                    {PCMHldRegB & {8{SelPCMB}}} |
                    {PCMHldRegC & {8{SelPCMC}}} |
                    {PCMHldRegD & {8{SelPCMD}}};

endmodule // tustint
```

부록 E

```
//
// tusrmictl.v :   Up-stream RAM output control
//
//
//
//
//
// Functional description:
//
//
//
// synthesis constraints
//
// sc   ClkList = {Clk19}
// sc   create_clock Clk19 -period 40 -waveform {0 20}
// sc   set_clock_skew -uncertainty 3 ClkList
// sc   set_fix_hold ClkList
// sc   set_false_path -from {fRESET_}
// sc   set_dont_touch_network ClkList
// sc   set_dont_touch_network UsClk19b
// sc   set_load 32 * load_of(m5c/CMDF00/D) UsClk19b

module tusrmictl
    (
    fRESET_,
    Clk19,
    UsClk19b,
    TxD,
    TxDVld,
    tETxCellType,
    tTxCellType,
    ETxSOH,
    ETxSFCnt,
    ETxGrpCnt,
    ETxDS0Cnt,
    tTxSOH,
    TxCellCnt,
    TxGrpCnt,
    TxDS0Cnt,
    TxPcmHit,
    StBlkSz0,
    StBlkSz1,
    UsPDOut,
    UsPRAdd,
    UsPREn,
    UsSigDOut,
    UsSigRAdd,
    UsSigREn,
    DlyDOut,
    DlyWAdd,
    DlyWEn,
    DlyDIn,
    DlyRAdd,
    DlyREn,
    UsSigRdBlk,
    ETxFS
    );

input           fRESET_;
input           Clk19;
input           tETxCellType;   //Early transmit cell type
input           tTxCellType;    //Transmit cell type
input           tTxSOH;         //Up-stream SONET overhead enable
input           ETxSOH;         //Early up-stream SONET overhead enable
input   [2:0]   ETxSFCnt;       //Early up-stream super frame count
input   [2:0]   ETxGrpCnt;      //Early POTS group number within the current
                                //up-stream TDM cell
input   [3:0]   ETxDS0Cnt;      //Early DS0 number within the current
                                //up-stream POTS group
input   [5:0]   TxCellCnt;      //Up-stream SPE cell number
input   [2:0]   TxGrpCnt;       //POTS group number within the current
                                //up-stream TDM cell
input   [3:0]   TxDS0Cnt;       //DS0 number within the current
                                //up-stream POTS group
input   [31:0]  TxPcmHit;       //up-stream hit signals from CAM
input   [31:0]  StBlkSz0;       //Bit 0 of all stored CAM block size counts
input   [31:0]  StBlkSz1;       //Bit 1 of all stored CAM block size counts
input           UsSigRdBlk;     //
input   [7:0]   UsPDOut;        //Up-stream PCM RAM data-out
input   [8:0]   DlyDOut;        //Delay RAM data-out
input   [7:0]   UsSigDOut;      //Up-stream signalling RAM data-out

output          UsClk19b;       //
output  [7:0]   TxD;            //Up-stream SONET frame data
output          TxDVld;         //Up-stream SONET valid frame data signal
output  [7:0]   UsPRAdd;        //Up-stream PCM RAM address-in
output  [4:0]   DlyRAdd;        //Delay RAM read address
output  [4:0]   DlyWAdd;        //Delay RAM write address
output          DlyREn;         //Delay RAM read enable
output          DlyWEn;         //Delay RAM write enable
output  [8:0]   DlyDIn;         //Delay RAM data-in
output          UsPREn;         //Up-stream PCM RAM read enable
output  [6:0]   UsSigRAdd;      //Up-stream Signalling RAM read address
output          UsSigREn;       //Up-stream Signalling RAM read enable
output          ETxFS;          //Early transmitt frame sync output

// local declarations

integer         i;              //

reg             Hit;            //
reg     [4:0]   HitAdd;         //
reg             AddSt;          //
reg             dETxSOH;        //
reg             nUsPREn;        //
reg             UsPREn;         //
reg             ValidGrp;       //
reg             SigFlg;         //
reg             DlyWEn;         //
reg             DlyREn;         //
reg             TxSigDSel;      //
reg             ETxFS;          //
reg     [4:0]   StHitAdd;       //
reg     [5:0]   StPRAdd;        //
reg     [1:0]   StBlkSz;        //
reg     [1:0]   BlkCntr;        //
reg     [1:0]   nBlkCntr;       //
reg     [1:0]   UsPRAddLwr;     //
reg     [2:0]   dETxGrpCnt;     //
reg     [3:0]   dETxDS0Cnt;     //
reg     [1:0]   SigWAdd;        //
reg     [6:0]   UsSigRAdd;      //
reg     [7:0]   SigReg0;        //
reg     [7:0]   SigReg1;        //
reg     [7:0]   SigReg2;        //
reg     [8:0]   DlyDIn;         //
reg     [1:0]   ETdmCellDly;    //
reg     [4:0]   DlyWAdd;        //
reg     [4:0]   DlyRAdd;        //
reg     [7:0]   TxD;            //
reg             TxDVld;         //
reg             SigFlgR0;       //
```

```
 reg             SigFlgR1;       //
 reg             SigFlgR2;       //
 reg             TxDEn;          //
 reg             dDlyWen;        //

 wire            nAddSt;         //
 wire            BlkCntr0;       //
 wire            BlkSz0;         //
 wire            Busy;           //
 wire            pValidGrp;      //
 wire            SigHit;         //
 wire            UsSigRen;       //
 wire            WeSigReg0;      //
 wire            WeSigReg1;      //
 wire            WeSigReg2;      //
 wire            nTxSig0Sel;     //
 wire            SigReg0Hit;     //
 wire            SigReg1Hit;     //
 wire            SigReg2Hit;     //
 wire    [1:0]   BlkSz;          //
 wire    [7:0]   UsPRAdd;        //
 wire    [7:0]   TxSig0;         //
 wire            SigFlgRAllClr;  //
 wire            SigFlgR0Clr;    //
 wire            SigFlgR1Clr;    //
 wire            SigFlgR2Clr;    //
 wire            UsClk19b;       //
 wire            nTxDVld;        //
 wire            TxSfg0Sel0;     //

 ////////////////////////////////////////////////////////////////////////////////
 // Parameters
 ////////////////////////////////////////////////////////////////////////////////

 parameter   DlyOffset       = 5'd8,
             TxFSCellNum     = 6'd40,
             TxFSGrpNum      = 3'd3,
             TxFSDS0Num      = 4'd1;

 ////////////////////////////////////////////////////////////////////////////////
 ////////////////////////////////////////////////////////////////////////////////
 //// Common stuff
 ////////////////////////////////////////////////////////////////////////////////
 ////////////////////////////////////////////////////////////////////////////////

 ////////////////////////////////////////////////////////////////////////////////
 // Clock inversion for up-stream signalling and PCM RAMs
 ////////////////////////////////////////////////////////////////////////////////

 assign UsClk19b = ~Clk19;

 ////////////////////////////////////////////////////////////////////////////////
 // Priority encoder with error signal for more than one CAM location hit
 ////////////////////////////////////////////////////////////////////////////////

 always @ ( TxPcmHit )
         begin
             Hit = 1'b0;
             HitAdd = 5'b00000;
             for (i = 31; i >= 0; i = i -1)
                 begin
                 if (TxPcmHit[i])
                     begin
                     Hit = 1'b1;
                     HitAdd = i;
                     end
                 end
         end

 ////////////////////////////////////////////////////////////////////////////////
 // Hit address latch           Latches address on CAM hit
 ////////////////////////////////////////////////////////////////////////////////

 always @ (posedge Clk19 or negedge !RESET_)
     begin
     if (!!RESET_)
         begin
         StHitAdd <= 5'b00000;
         end
     else if (Hit & ~Busy)
             begin
             StHitAdd <= HitAdd;
             end
     end

 ////////////////////////////////////////////////////////////////////////////////
 // RAM address latch
 ////////////////////////////////////////////////////////////////////////////////

 always @ (posedge Clk19 or negedge !RESET_)
     begin
     if (!!RESET_)
         begin
         StPRAdd <= 6'b000000;
         StBlkSz <= 2'b00;
         end
     else if (AddSt)
             begin
             StPRAdd <= {ETxSFCnt[0],StHitAdd};
             StBlkSz <= BlkSz;
             end
     end

 ////////////////////////////////////////////////////////////////////////////////
 // Block size multiplexer to retrive the stored block size of the hit CAM
 // location
 ////////////////////////////////////////////////////////////////////////////////

 assign BlkSz = (StBlkSz1[StHitAdd],StBlkSz0[StHitAdd]);

 ////////////////////////////////////////////////////////////////////////////////
 ////////////////////////////////////////////////////////////////////////////////
 //// PCM RAM address and data control
 ////////////////////////////////////////////////////////////////////////////////
 ////////////////////////////////////////////////////////////////////////////////

 ////////////////////////////////////////////////////////////////////////////////
 // Delayed Hit signal used as a strobe to store the CAM hit address, as a start
 // signal for the BlkCntr, and as a RAM read enable for the first DS0.
 ////////////////////////////////////////////////////////////////////////////////

 assign nAddSt = Hit & ~Busy;

 always @ (posedge Clk19 or negedge !RESET_)
     begin
     if (!!RESET_)
         begin
         AddSt <= 1'b0;
         dETxSOH <= 1'b0;
         end
     else
         begin
```

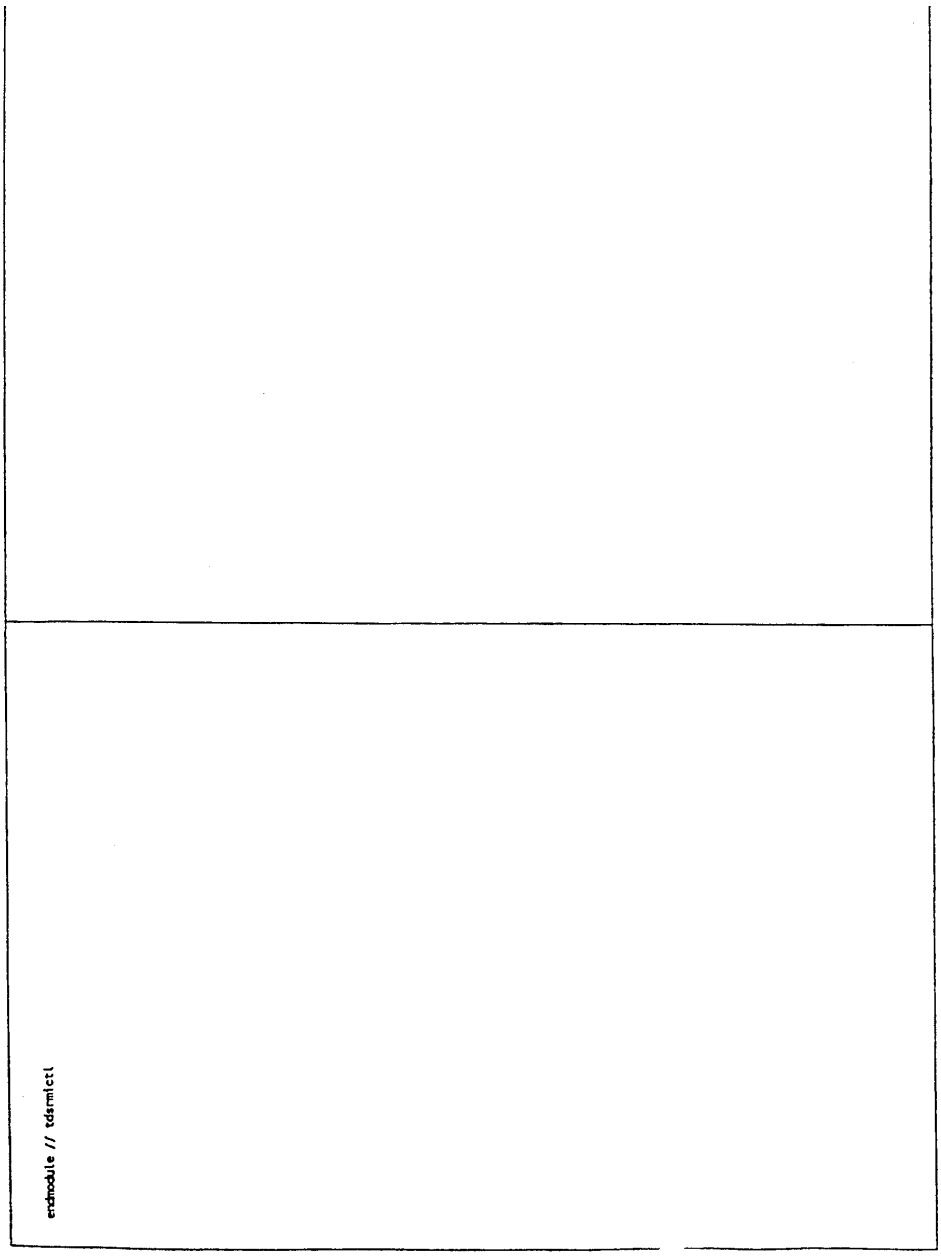

등록특허 10-0507023

```
        AddSt <= nAddSt;
        dETxSOH <= ETxSOH;
        end
    end

////////////////////////////////////////////////////////////////////////////////
// Byte counter and control
////////////////////////////////////////////////////////////////////////////////

assign BlkCntr0 = (BlkCntr == 2'b00);
assign BlkSz0 = (BlkSz == 2'b00);

always @ (posedge Clk19 or negedge iRESET_)
    begin
    if (!iRESET_)
        BlkCntr <= 2'b00;
    else
        BlkCntr <= nBlkCntr;
    end

always @ (AddSt or StBlkSz or BlkSz0 or dETxSOH or BlkCntr0 or BlkCntr)
    begin
    if (AddSt & ~BlkSz0)
        nBlkCntr = 2'b01;
    else if (!dETxSOH)
            if ((BlkCntr >= StBlkSz) || BlkCntr0)
                nBlkCntr = 2'b00;
            else
                nBlkCntr = BlkCntr + 1'b1;
        else
            nBlkCntr = BlkCntr;
    end

assign Busy = nBlkCntr != 2'b00;

////////////////////////////////////////////////////////////////////////////////
// PCM RAM read address and read enable
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk19 or negedge iRESET_)
    begin
    if (!iRESET_)
        begin
        UsPRAddLwr <= 2'b00;
        nUsPRen <= 1'b0;
        UsPRen <= 1'b0;
        end
    else
        begin
        UsPRAddLwr <= BlkCntr;
        nUsPRen <= nAddSt | ~ETxSOH & ~(nBlkCntr == 2'b00);
        UsPRen <= nUsPRen;
        end
    end

assign UsPRAdd = {StPRAdd,UsPRAddLwr};


////////////////////////////////////////////////////////////////////////////////
// Signalling RAM address and data control
////////////////////////////////////////////////////////////////////////////////

assign pValidGrp = tETxCellType & (ETxGrpCnt != 3'b111);

always @ (posedge Clk19 or negedge iRESET_)
    begin
    if (!iRESET_)
        begin
        dETxGrpCnt <= 3'b000;
        dETxDS0Cnt <= 4'b0000;
        ValidGrp <= 1'b0;
        end
    else
        begin
        dETxGrpCnt <= ETxGrpCnt;
        dETxDS0Cnt <= ETxDS0Cnt;
        ValidGrp <= pValidGrp;
        end
    end

assign SigHit = ~dETxSOH & ValidGrp & ((1'b0,ETxSFCnt) == dETxDS0Cnt) &
                    (nUsPRen);

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        SigWAdd <= 2'b00;
    else if (SigHit)
        begin
        if (dETxGrpCnt == 3'd0 | dETxGrpCnt == 3'd3) SigWAdd <= 2'b00;
        if (dETxGrpCnt == 3'd1 | dETxGrpCnt == 3'd4) SigWAdd <= 2'b01;
        if (dETxGrpCnt == 3'd2 | dETxGrpCnt == 3'd5) SigWAdd <= 2'b10;
        end

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        begin
        UsSigRAdd <= 7'h00;
        end
    else if (SigHit)
        begin
        UsSigRAdd <= (((AddSt) ? StHitAdd : StPRAdd[4:0]),BlkCntr);
        end

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        SigFlg <= 1'b0;
    else if (SigFlg & UsSigRen)
            SigFlg <= 1'b0;
        else if (~SigFlg & SigHit)
                SigFlg <= 1'b1;

assign UsSigRen = SigFlg & ~UsSigRdBlk;

assign WeSigReg0 = UsSigRen & SigWAdd == 2'b00;
assign WeSigReg1 = UsSigRen & SigWAdd == 2'b01;
assign WeSigReg2 = UsSigRen & SigWAdd == 2'b10;

assign SigFlgRAllClr = ~(ETdmCellDly[0] | DlyRen);

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        SigFlgR0 <= 1'b0;
    else if (SigFlgR0 & (SigFlgR0Clr | SigFlgRAllClr))
            SigFlgR0 <= 1'b0;
        else if (~SigFlgR0 & WeSigReg0)
                SigFlgR0 <= 1'b1;

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        SigFlgR1 <= 1'b0;
    else if (SigFlgR1 & (SigFlgR1Clr | SigFlgRAllClr))
            SigFlgR1 <= 1'b0;
```

```
        else if (~SigFlgR1 & WeSigReg1)
                SigFlgR1 <= 1'b1;

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        SigFlgR2 <= 1'b0;
    else if (SigFlgR2 & (SigFlgR2Clr | SigFlgRAllClr))
            SigFlgR2 <= 1'b0;
        else if (~SigFlgR2 & WeSigReg2)
                SigFlgR2 <= 1'b1;

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        SigReg0 <= 8'h00;
    else if (WeSigReg0)
        SigReg0 <= UsSig0Out;

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        SigReg1 <= 8'h00;
    else if (WeSigReg1)
        SigReg1 <= UsSig0Out;

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        SigReg2 <= 8'h00;
    else if (WeSigReg2)
        SigReg2 <= UsSig0Out;

////////////////////////////////////////////////////////////////////////////////
// Delay RAM write and read address and control signals and final TxD data
// muxing and retiming
////////////////////////////////////////////////////////////////////////////////


always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        begin
        DlyDIn <= 9'd0;
        end
    else
        begin
        DlyDIn <= {UsPRen,((UsPRen) ? UsPDOut : 8'h00)};
        end

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        begin
        ETdmCellDly <= 2'b00;
        DlyWen <= 1'b0;
        dDlyWen <= 1'b0;
        DlyWAdd <= 5'b00000;
        DlyRAdd <= 5'b00000;
        DlyRen <= 1'b0;
        TxSigDSel <= 1'b0;
        end
    else
        begin
        ETdmCellDly <= {ETdmCellDly[0],tETxCellType};
        DlyWen <= ETdmCellDly[1];
        dDlyWen <= DlyWen;
        DlyWAdd <= DlyWAdd + 1'b1;
        DlyRAdd <= DlyWAdd + DlyOffset;
        DlyRen <= tTxCellType;
        TxSigDSel <= nTxSigDSel;
        end
```

```
assign nTxSigDSel = tTxCellType & ~tTxSOH & (TxDSOCnt == 4'b1000);

assign SigReg0Hit = TxGrpCnt == 3'd0 | TxGrpCnt == 3'd3;
assign SigReg1Hit = TxGrpCnt == 3'd1 | TxGrpCnt == 3'd4;
assign SigReg2Hit = TxGrpCnt == 3'd2 | TxGrpCnt == 3'd5;

assign SigFlgR0Clr = TxSigDSel & SigReg0Hit;
assign SigFlgR1Clr = TxSigDSel & SigReg1Hit;
assign SigFlgR2Clr = TxSigDSel & SigReg2Hit;

assign TxSigD = (SigReg0 & {8{(SigReg0Hit & SigFlgR0)}}) |
                (SigReg1 & {8{(SigReg1Hit & SigFlgR1)}}) |
                (SigReg2 & {8{(SigReg2Hit & SigFlgR2)}});

assign nTxDVld = DlyDOut[8]
                 SigFlgR0 & SigFlgR0Clr
                 SigFlgR1 & SigFlgR1Clr
                 SigFlgR2 & SigFlgR2Clr;

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        begin
        TxDEn <= 1'b0;
        end
    else if (~DlyWen & dDlyWen)
        begin
        TxDEn <= 1'b1;
        end

assign TxSigDSelQ = TxSigDSel & ~DlyDOut[8];

always @ (posedge Clk19 or negedge iRESET_)
    if (!iRESET_)
        begin
        TxDVld <= 1'b0;
        TxD <= 8'h00;
        end
    else
        begin
        TxDVld <= (DlyRen & TxDEn) ? nTxDVld : 1'b0;
        TxD <= (DlyRen & TxDEn) ? ((TxSigDSelQ) ? TxSigD: DlyDOut[7:0]) : 8'h00;
        end

////////////////////////////////////////////////////////////////////////////////
// Generation of early frame sync signal
////////////////////////////////////////////////////////////////////////////////

always @ (posedge Clk19 or negedge iRESET_)
    begin
    if (!iRESET_)
        begin
        ETxFS <= 1'b0;
        end
    else
        begin
        ETxFS <= TxCellCnt == TxFSCellNum &
                 TxGrpCnt == TxFSGrpNum &
                 TxDSOCnt == TxFSDSONum;
        end
    end

endmodule // tuarmictl
```

Appendix E page 4



(57) 청구의 범위

청구항 1.

삭제

청구항 2.

삭제

청구항 3.

다수의 통신디바이스 세트에 대한 신호를 가지는 신호 스트림을 전송하는 통신 시스템에서, 상기 신호 스트림으로부터 제 1 통신디바이스 세트에 대한 신호를 추출하는 방법에 있어서:

(a) 제 1 통신디바이스 세트에 대한 신호 세트를 적어도 하나이상 포함하는 신호 세트의, 상기 신호 스트림내에서의 위치 식별정보를 저장하는 단계;

(b) 수신된 신호 스트림에서 수신된 신호 세트의 위치식별정보를 다수의 상기 저장된 위치식별정보와 동시에 비교하는 단계;

(c) 제 1 신호 세트의 위치식별정보가 제 1 통신디바이스 세트에 대한 신호 세트의 상기 저장된 위치식별정보와 매치할 때, 상기 제 1 신호 세트를 상기 신호 스트림로부터 추출하는 단계;

(d) 메모리 어드레스 및 제어신호를 발생시키는 단계를 포함하는, 라우팅 신호를 발생시키는 단계; 및

(e) 상기 추출된 제 1 신호 세트를 상기 라우팅 신호에 기초하여 상기 제 1 통신디바이스 세트로 라우팅하는 단계;를 포함하고 있으며, 상기 라우팅하는 단계는 메모리에 상기 추출된 제 1 신호 세트를 저장하기 위해, 상기 발생된 메모리 신호를 사용하는 단계를 포함하는 것을 특징으로 하는 신호추출 방법.

청구항 4.

제 3 항에 있어서, 상기 라우팅 단계는, 제 1 신호 세트를 추출하기 위해 상기 메모리를 판독하는 단계, 및 상기 판독된 제 1 신호 세트를 제 1 통신디바이스 세트로 전달하는 단계를 더 포함하는 것을 특징으로 하는 신호추출 방법.

청구항 5.

서로 다른 통신애플리케이션에 대한 신호를 가지는 신호 스트림을 전송하는 통신 시스템에서, 수신된 신호 스트림으로부터 제 1 형태의 통신애플리케이션에 대한 신호를 추출하는 방법에 있어서:

(a) 상기 수신된 신호 스트림에서, 제 1 형태의 애플리케이션에 대한 신호 세트의 위치를 식별하는 위치식별신호를 저장하는 단계;

(b) 상기 수신된 신호 스트림에서 각각의 수신된 신호 세트에 대한 위치식별 신호를 발생시키는 단계;

(c) 상기 각각의 수신된 신호 세트에 대한 상기 발생된 위치식별 신호를 다수의 저장된 위치식별신호와 동시에 비교하는 단계;

(d) 상기 발생된 위치식별신호 중의 하나가 상기 저장된 위치식별신호 중의 하나와 매치할 때, 상기 발생된 위치식별 신호에 대응하는 신호 세트를 상기 신호 스트림으로부터 추출하는 단계;

(e) 메모리 어드레스 및 제어 신호를 발생시키는 단계를 포함하는, 라우팅 신호를 발생시키는 단계; 및

(f) 상기 추출된 제 1 신호 세트를 상기 라우팅 신호에 기초하여 상기 제 1 통신디바이스 세트로 라우팅하는 단계;를 포함하고 있으며, 상기 라우팅하는 단계는 메모리에 상기 추출된 제 1 신호 세트를 저장하기 위해, 상기 발생된 메모리 신호를 사용하는 단계를 포함하는 것을 특징으로 하는 신호추출 방법.

청구항 6.

다수의 통신디바이스 세트에 대한 신호를 가지는 신호 스트림을 전송하는 통신 시스템에서, 제 1 통신디바이스 세트에 대한 신호를 상기 신호 스트림에 삽입하는 방법에 있어서:

(a) 제 1 통신디바이스 세트로부터의 전송을 위한 각각의 신호 세트에 대해서, 상기 신호 스트림에서의 신호 세트의 위치를 식별하는 포인터를 저장하는 단계;

(b) 신호 스트림에서의 신호 세트의 위치를 식별하는 포인터를 발생시키는 단계;

(c) 각각 발생된 포인터를 다수의 상기 저장된 포인터와 동시에 비교하는 단계; 및

(d) 발생된 포인터가 저장된 포인터와 매치할 때, 메모리 어드레스 및 제어 신호를 발생시키고, 상기 매치된 포인터에 대응하는 신호세트를 메모리로부터 검색하기 위해 상기 발생된 메모리 신호를 사용하고, 상기 매치된 저장된 포인터에 대응하는 신호 세트를 상기 신호 스트림에 삽입하는 단계;를 포함하는 것을 특징으로 하는 신호삽입 방법.

청구항 7.

다수의 통신디바이스 세트에 대한 신호를 가지는 신호 스트림을 전송하는 통신 시스템에서, 제 1 통신디바이스 세트에 대한 신호를 상기 신호 스트림에 삽입하는 방법에 있어서:

(a) 제 1 통신디바이스 세트로부터의 전송을 위한 각각의 신호 세트에 대해서, 상기 신호 세트의 전송에 대한 시간주기를 식별하는 포인터를 저장하는 단계;

(b) 상기 신호 스트림을 전송하는 동안, 신호 세트의 전송에 대한 시간주기를 식별하는 포인터를 발생시키는 단계;

(c) 각각의 발생된 포인터를 다수의 상기 저장된 포인터와 동시에 비교하는 단계; 및

(d) 발생된 포인터가 저장된 포인터와 매치할 때, 메모리 어드레스 및 제어 신호를 발생시키고, 상기 매치된 포인터에 대응하는, 상기 제1 통신 디바이스 세트로부터의 신호세트를 메모리로부터 검색하기 위해 상기 발생된 메모리 신호를 사용하고, 상기 매치된 저장된 포인터에 대응하는 신호 세트를 상기 신호 스트림에 삽입하는 단계;를 포함하는 것을 특징으로 하는 신호삽입 방법.

청구항 8.

신호 스트림으로부터 추출될 필요가 있는 신호 세트를 식별하기 위해, 상기 신호 스트림에서 신호 세트에 대한 위치식별 정보를 저장하는 내용주소화 메모리, 및 라우팅 신호를 발생시키고, 상기 추출된 신호 세트를 상기 라우팅 신호에 기초하여 제1 통신 디바이스 세트로 라우팅하는 메모리 인터페이스를 사용하는 단계를 포함하고 있으며, 상기 라우팅 신호를 발생시키는 것은 메모리 어드레스 및 제어신호를 발생시키는 것을 포함하고, 상기 라우팅은 상기 추출된 신호를 메모리에 저장하기 위해 상기 발생된 메모리 신호를 사용하는 것을 포함하는 것을 특징으로 하는 시분할 디멀티플렉싱 방법.

청구항 9.

신호 세트를 신호 스트림로 삽입하기 위한 시간주기를 식별하기 위한 내용주소화 메모리, 및 메모리 어드레스 및 제어신호를 발생시키고, 상기 신호 스트림에 삽입되어야 할 신호 세트를 메모리로부터 검색하기 위해 상기 발생된 메모리 신호를 사용하는 메모리 인터페이스를 사용하는 단계를 포함하는 것을 특징으로 하는 시분할 멀티플렉싱 방법.

청구항 10.

광섬유 원격통신 네트워크에서, 텔레포니 애플리케이션에 대한 신호 및 넌-텔레포니 애플리케이션에 대한 신호를 가지는 신호 스트림을 수신하며, 수신된 신호 스트림에서 텔레포니 애플리케이션에 대한 신호의 위치식별정보를 저장하는 내용주소화 메모리, 및 상기 내용주소화 메모리에 결합되어 있고, 메모리 어드레스 및 제어신호를 발생시키고, 상기 수신된 신호 스트림으로부터 추출된 신호를 메모리에 저장하기 위해 상기 발생된 메모리 신호를 사용하는 메모리 인터페이스를 포함하는 것을 특징으로 하는 광 네트워크 유닛.

청구항 11.

다수의 통신디바이스 세트에 대한 신호를 가지는 신호 스트림을 전송하는 통신 시스템에서, 상기 신호 스트림으로부터 제 1 통신디바이스 세트에 대한 신호를 추출하는 장치에 있어서:

(a) 제 1 통신디바이스 세트에 대한 신호 세트의, 신호 스트림내에서의 위치를 식별하는 위치식별신호를 저장하기 위한 메모리; 및

(b) 상기 메모리에 결합되어 있으며, 각각의 수신된 신호 세트에 대해서 수신된 신호 세트의 위치를 식별하는 위치식별신호를 수신하고, 상기 각각의 수신된 위치식별신호를 다수의 상기 저장된 위치식별신호와 동시에 비교하는 비교기; 및

(c) 상기 비교기에 결합되어 있으며, 상기 비교기로부터의 신호에 응답하여, 상기 신호 스트림으로부터 제1 신호 세트를 추출하고, 상기 추출된 신호를 메모리에 저장하기 위해 어드레스 및 제어신호를 발생시키는 메모리 인터페이스;를 포함하고 있으며,

상기 비교기는, 제 1 신호 세트의 위치식별신호가 저장된 위치식별 신호와 매치할 때, 제1 신호 세트가 상기 신호 스트림로부터 추출되어야 함을 지시하는 신호를 발생시키는 것을 특징으로 하는 신호추출 장치.

청구항 12.

다수의 통신디바이스 세트에 대한 신호를 가지는 신호 스트림을 전송하는 통신 시스템에서, 제 1 통신디바이스 세트로부터의 신호를 상기 신호 스트림에 삽입하는 장치에 있어서:

(a) 제 1 통신디바이스 세트로부터의 신호 세트에 대한, 상기 신호 스트림내에서의 위치를 식별하는 위치식별신호를 저장하기 위한 메모리; 및

(b) 상기 메모리에 결합되어 있으며, 신호 스트림이 전송되는 동안 상기 신호 스트림에서 신호 세트의 위치를 식별하는 위치식별신호를 수신하고, 각각의 수신된 위치식별 신호를 다수의 상기 저장된 위치식별신호와 동시에 비교하는 비교기; 및

(c) 상기 비교기에 결합되어 있으며, 상기 비교기로부터의 신호에 응답하여, 메모리 어드레스 및 제어신호를 발생시키고, 상기 제1 신호 세트를 저장하는 메모리로부터 상기 제1 신호 세트를 검색하기 위해 상기 메모리 신호를 사용하는 메모리 인터페이스;를 포함하고 있으며,

상기 비교기는, 위치식별 신호가 제 1 신호 세트의 저장된 위치식별신호와 매치할 때, 제 1 신호 세트가 신호 스트림에 삽입되어야 함을 지시하는 신호를 발생시키는 것을 특징으로 하는 신호삽입 장치.

청구항 13.

신호 스트림으로부터 추출될 필요가 있는 신호 세트를 식별하기 위해, 상기 신호 스트림에서 신호 세트의 위치식별정보를 저장하는 내용주소화 메모리, 및 라우팅 신호를 발생시키고, 상기 추출된 신호 세트를 상기 라우팅 신호에 기초하여 제 1 통신 디바이스 세트로 라우팅하는 메모리 인터페이스를 포함하고 있으며, 상기 라우팅 신호를 발생시키는 것은 메모리 어드레스 및 제어신호를 발생시키는 것을 포함하고, 상기 라우팅은 상기 추출된 신호를 메모리에 저장하기 위해 상기 발생된 메모리 신호를 사용하는 것을 포함하는 것을 특징으로 하는 시분할 디멀티플렉서.

청구항 14.

신호 세트를 신호 스트림에 삽입하기 위한 시간주기를 식별하기 위한 내용주소화 메모리, 및 메모리 어드레스 및 제어신호를 발생시키고, 상기 신호 스트림에 삽입되어야 할 신호 세트를 메모리로부터 검색하기 위해 상기 발생된 메모리 신호를 사용하는 메모리 인터페이스를 포함하는 것을 특징으로 하는 시분할 멀티플렉서.

청구항 15.

제 5 항에 있어서, 상기 라우팅하는 단계는 상기 제 1 신호 세트를 추출하기 위해 메모리를 판독하는 단계 및 상기 판독된 제1 신호 세트를 제1 통신 디바이스 세트로 전달하는 단계를 더 포함하는 것을 특징으로 하는 신호추출 방법.

청구항 16.

제 15 항에 있어서, 상기 라우팅하는 단계는 메모리의 어드레스를 발생시키는 단계 및 상기 메모리로부터 상기 제 1 신호 세트를 추출하기 위해, 상기 메모리를 판독하기 위해 상기 어드레스를 사용하는 단계를 더 포함하는 것을 특징으로 하는 신호추출 방법.

청구항 17.

제 6 항에 있어서, 상기 메모리에 대한 어드레스를 발생시키고, 상기 메모리에 상기 신호 세트를 저장하기 위해 상기 어드레스를 사용하는 단계를 더 포함하는 것을 특징으로 하는 신호삽입 방법.

청구항 18.

제 17 항에 있어서, 상기 메모리로부터 검색된 신호 세트를 상기 신호 스트림에 삽입하기 전에 상기 메모리로부터 검색된 상기 신호 세트를 지연 버퍼에 저장하는 단계를 더 포함하는 것을 특징으로 하는 신호삽입 방법.

청구항 19.

제 7 항에 있어서, 상기 메모리로부터 검색된 신호 세트를 상기 신호 스트림에 삽입하기 전에 상기 메모리로부터 검색된 상기 신호 세트를 지연 버퍼에 저장하는 단계를 더 포함하는 것을 특징으로 하는 신호삽입 방법.

청구항 20.

제 19 항에 있어서, 상기 메모리에 대한 어드레스를 발생시키고, 상기 메모리에 상기 신호 세트를 저장하기 위해 상기 어드레스를 사용하는 단계를 더 포함하는 것을 특징으로 하는 신호삽입 방법.

청구항 21.

제 11 항에 있어서, 상기 제 1 신호 세트를 추출하기 위해 메모리를 판독하고, 상기 판독된 제 1 신호 세트를 상기 제 1 통신 디바이스 세트로 전달하는 인터페이스를 더 포함하는 것을 특징으로 하는 신호추출 장치.

청구항 22.

제 21 항에 있어서, 포인터 테이블이 상기 메모리의 어드레스를 발생시키도록 해주고, 상기 메모리로부터 상기 제 1 신호 세트를 추출하기 위해, 상기 메모리를 판독하기 위해 상기 어드레스를 상기 인터페이스에 제공하는 포인터 제어기를 더 포함하는 것을 특징으로 하는 신호추출 장치.

청구항 23.

제 12 항에 있어서, 상기 제 1 통신 디바이스 세트로부터의 상기 제 1 신호 세트를 상기 메모리로 라우팅하고, 상기 제 1 신호 세트를 상기 메모리에 기입하는 인터페이스를 더 포함하는 것을 특징으로 하는 신호삽입 장치.

청구항 24.

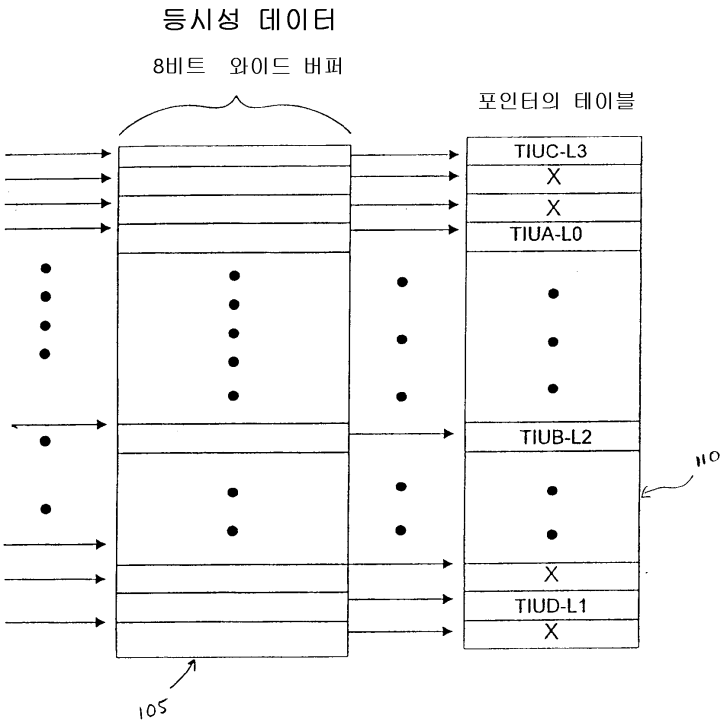
제 23 항에 있어서, 포인터 테이블이 상기 메모리의 어드레스를 발생시키도록 해주고, 상기 제 1 신호 세트를 상기 메모리로 라우팅하기 위해 상기 어드레스를 상기 인터페이스에 제공하는 포인터 제어기를 더 포함하는 것을 특징으로 하는 신호삽입 장치.

청구항 25.

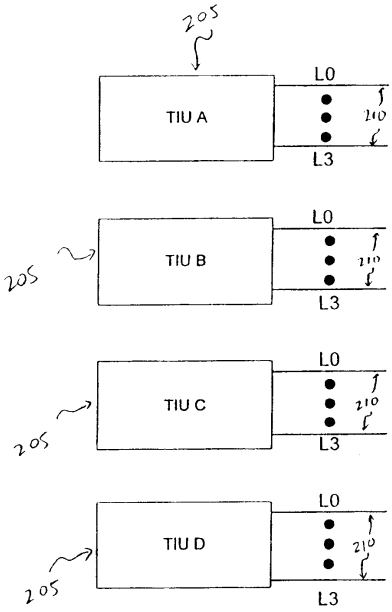
제 24 항에 있어서, 상기 제 1 신호 세트를 상기 신호 스트림에 삽입하기 전에 메모리로부터 검색된 제 1 신호 세트를 저장하는 지연 버퍼를 더 포함하는 것을 특징으로 하는 신호삽입 장치.

도면

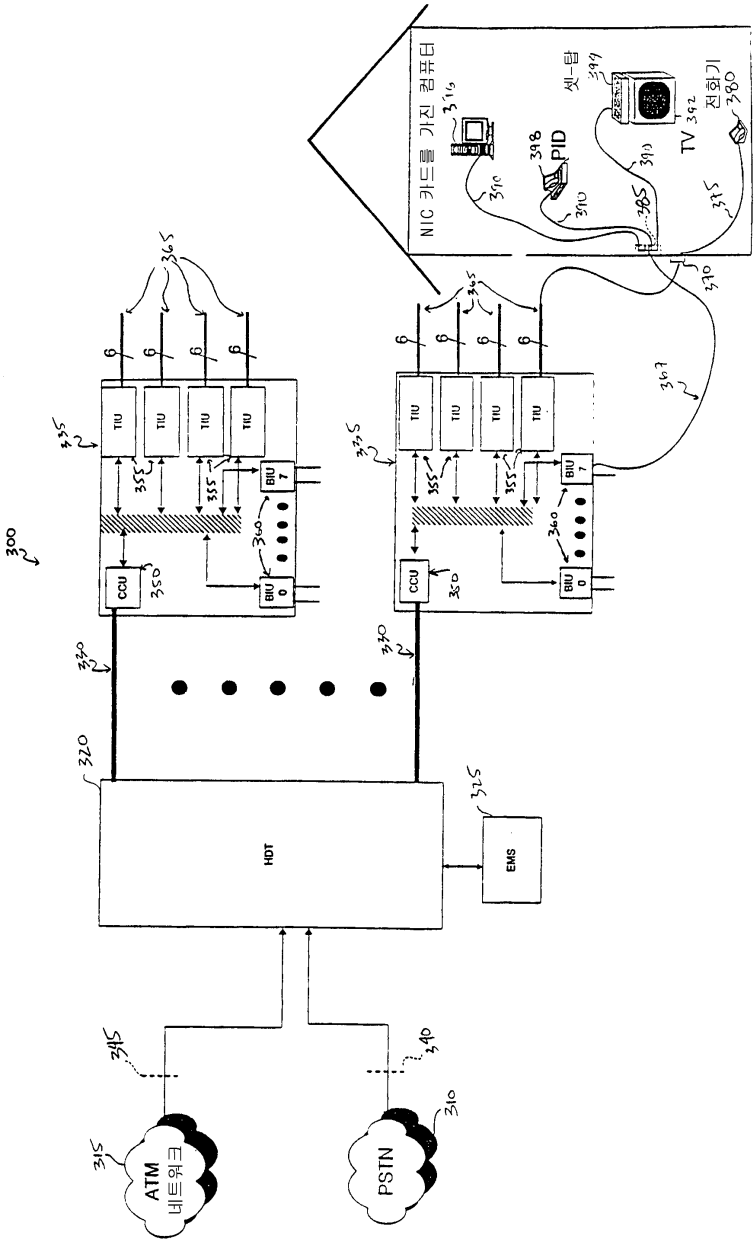
도면1



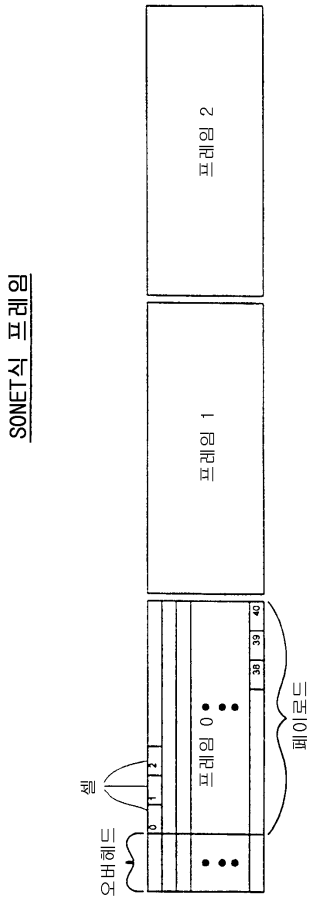
도면2



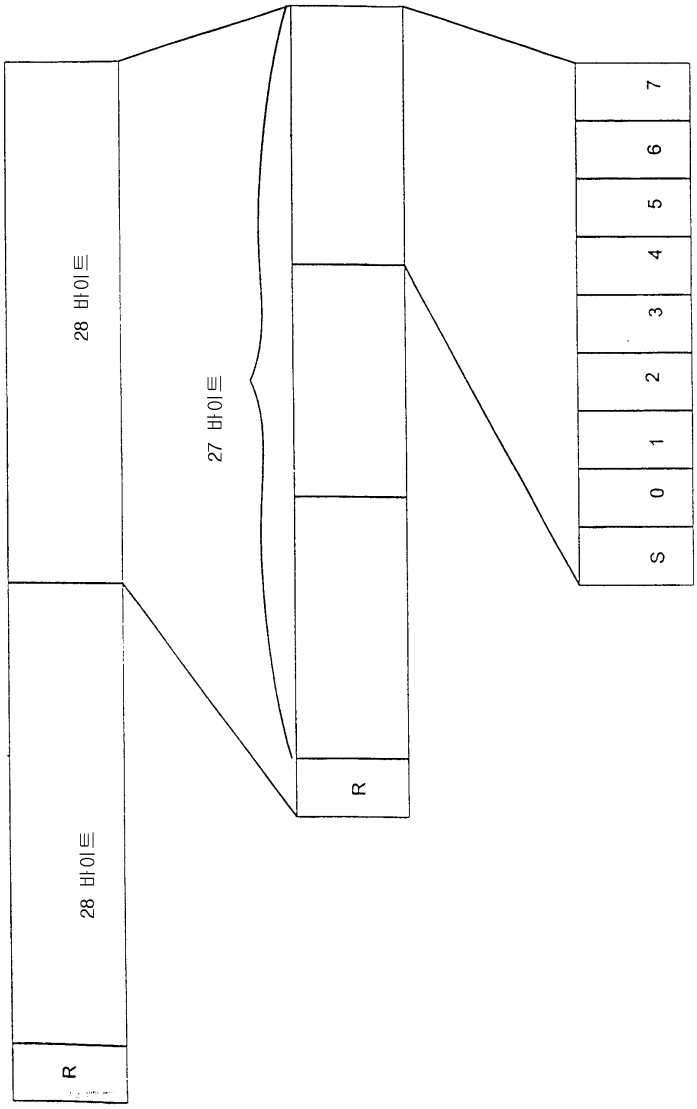
도면3


NIC 카드를 가진 컴퓨터
PID
셋-탑
TV
전화기
TIU
BIU 7
BIU 0
CCU
HDT
EMS
ATM 네트워크
PSTN

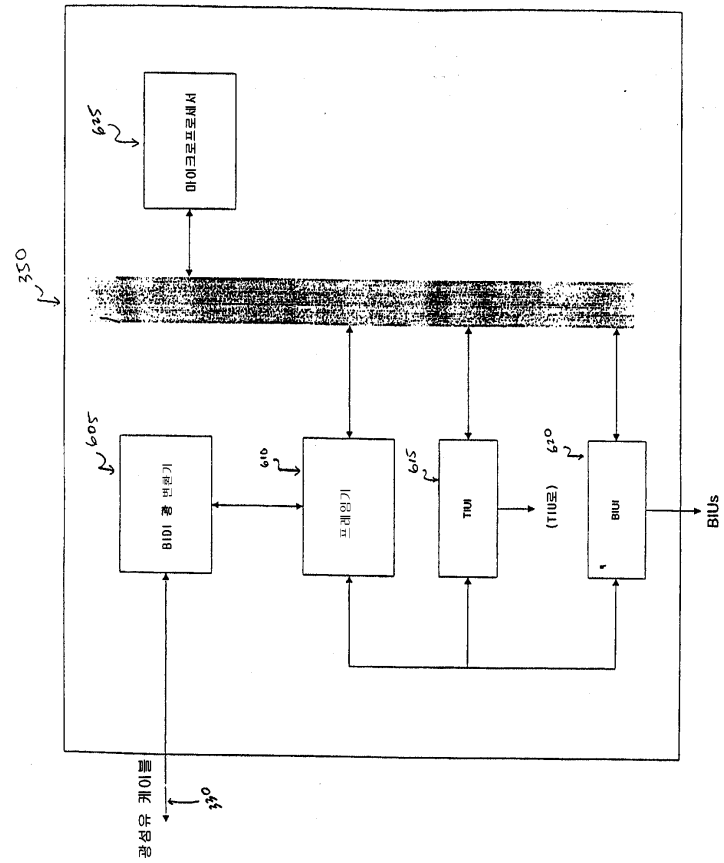
도면4



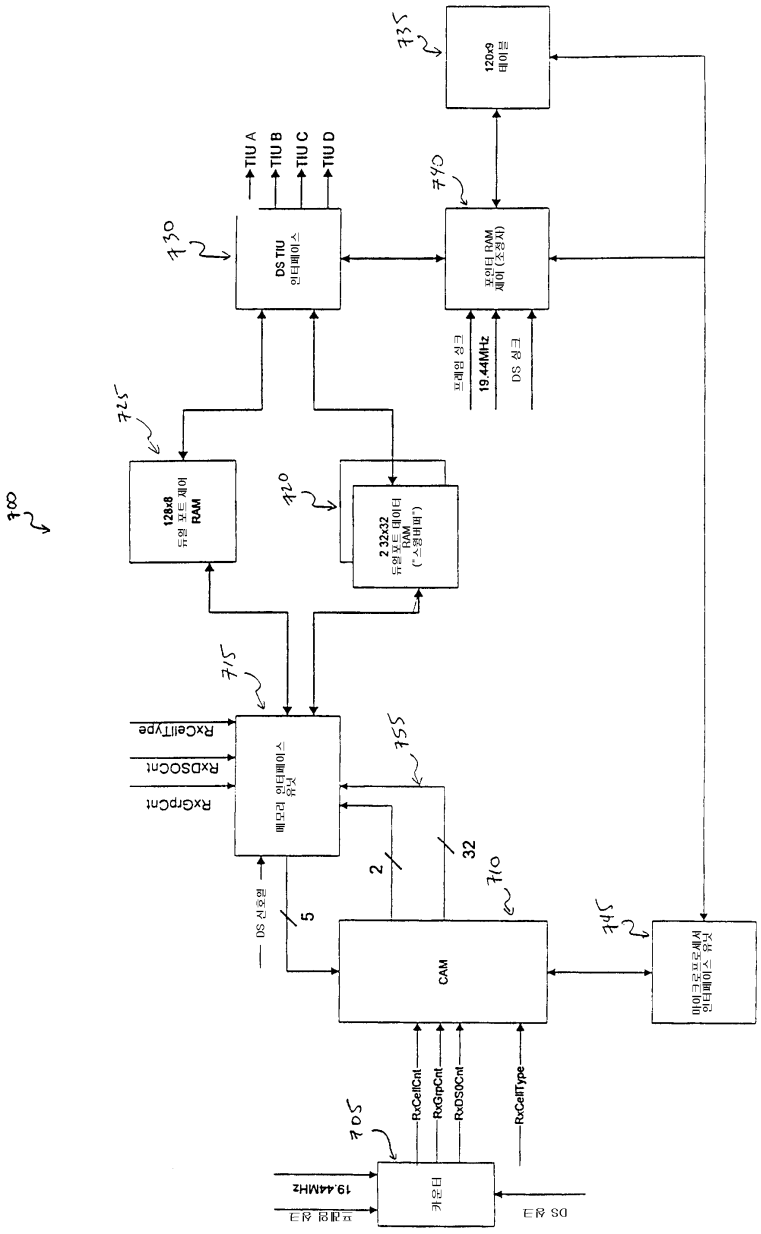
도면5



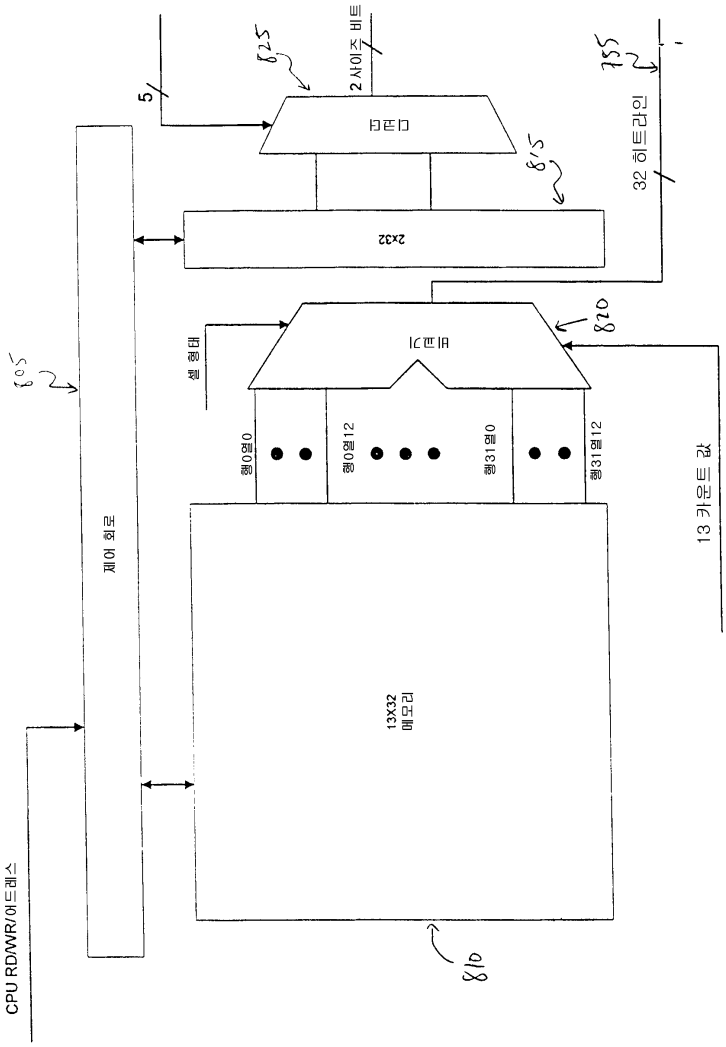
도면6



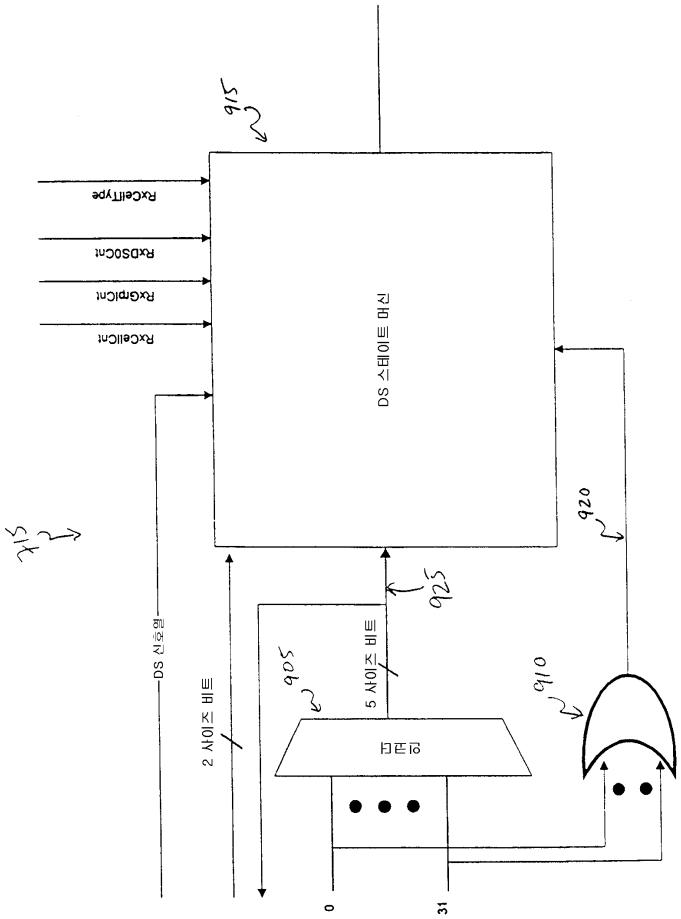
도면7


700
705
710
715
720
725
730
735
740
745
750
755
TIU A
TIU B
TIU C
TIU D
DS TIU 인터페이스
120x9 테이블
포인터 RAM 제어 (조정자)
128x8 듀얼 포트 제어 RAM
2 32x32 듀얼 포트 데이터 RAM (스위칭매트)
메모리 인터페이스 유닛
CAM
마이크로프로세서 인터페이스 유닛
RxCellCnt
RxGrpCnt
RxDSOCnt
RxCellType
19.44MHz
DS 신호
프레임 싱크
DS 싱크
5
2
32

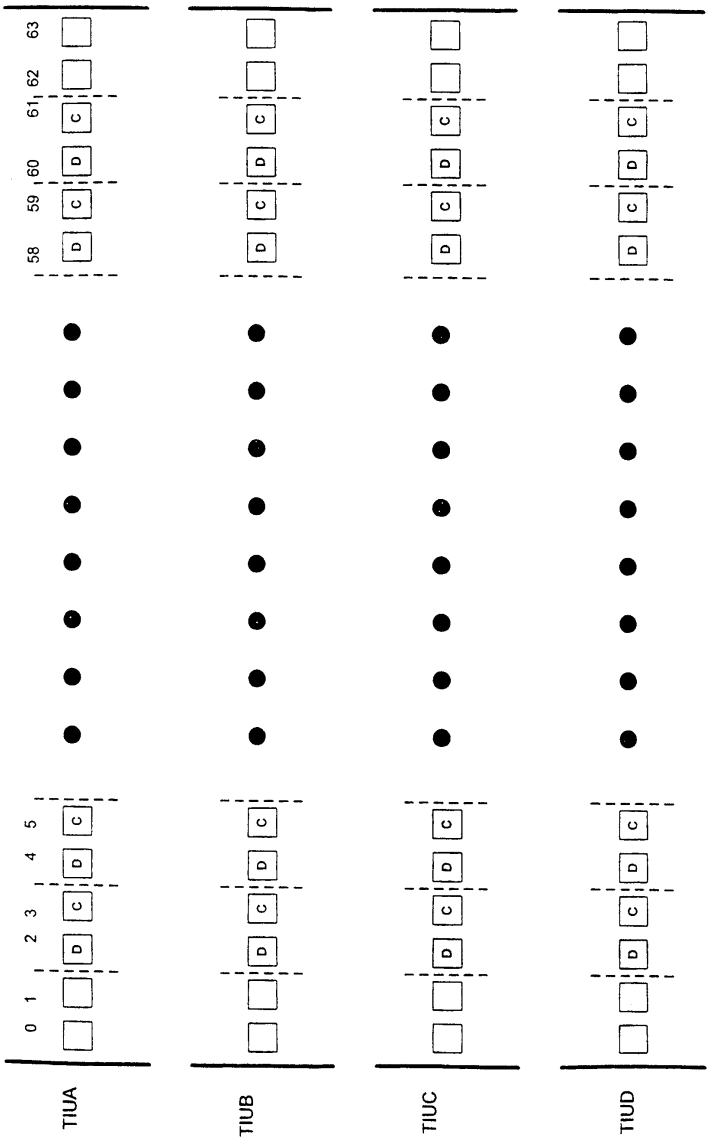
도면8


CPU RD/WR/어드레스
제어 회로
13X32 메모리
행0열0
행0열12
행31열0
행31열12
셀 형태
비교기
디코더
2x32
2 싸이즈 비트
32 히트라인
13 카운트 값

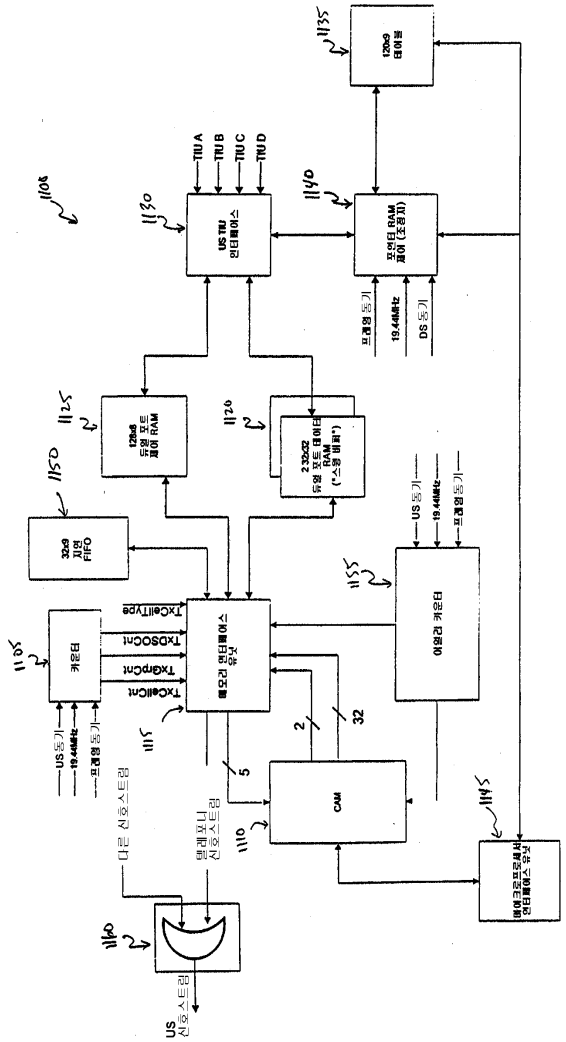
도면9


915
RxCellType
RxDS0Cnt
RxGrpICnt
RxCellCnt
DS 스테이트 머신
920
925
905
910
5 사이즈 비트
2 사이즈 비트
DS 신호들
디코딩
0
31

도면10



도면11


TXCellType
TxDSOCnt
TxGrpCnt
TxCellCnt
TxCellCnt
US동기
19.44MHz
프레임 동기
다른 신호스트림
탤레포니 신호스트림
US 신호스트림
DS동기
2
32
5