DOI: 10.3745/KIPSTD.2008.15-D.2.213

실시간 소프트웨어 분석 및 설계 기법을 이용한 뮤직 임베디드시스템 소프트웨어의 구현

최 성 민[†] · 오 훈[††]

요 약

기존의 뮤직 소프트웨어 설계에서는 실시간 태스킹 모델을 고려하지 않았기 때문에 주요 모듈의 실행 시간에 대한 시간적 제약을 고려하는 시스템의 성능 분석이 어려웠으며, 소프트웨어 구조가 복잡하고 확장성이 부족하였다. 이러한 문제를 종합적으로 해결하기 위하여 RTSA를 사용하여 뮤직 임베디드시스템을 분석하고 분석 사양에 DARTS를 적용하여 컨커런트 태스킹 아키텍처를 설계하였으며, 각 태스크가 갖는 시간적 제약을 만족될 수 있는지를 검증하기 위하여 RMA (Rate Monotonic Analysis)를 사용하여 시스템 성능을 분석하였다. 설계 결과는 임베디드리눅스를 탑재한 인텔 벌버드 (Bulverde) 마이크로프로세서 기반의 Xhyper272 보드에서 구현하였다.

키워드 : 뮤직 임베디드시스템, 실시간, 컨커런트 태스크, DARTS, 스케줄링

Implementation of Music Embedded System Software Using Real Time Software Analysis and Design Method

Seong Min Choi[†] · Hoon Oh[††]

ABSTRACT

The existing approaches for the music application have not considered a real-time multi-tasking model. So, it suffers from a high complexity and a low flexibility in design as well as lack of predictability for the timely execution of critical tasks. In this paper, we design a new concurrent tasking architecture for a real-time embedded music system and examine if all real-time tasks can finish execution within their respective time constraints. The design is implemented on the Linux based Xhyper272 Board that uses the Intel Bulverde microprocessor.

Key Words : Music Embedded System, Real-Time, Concurrent Tasks, DARTS, Scheduling

1. 서 론

마이크로프로세서의 가격이 저렴해짐에 따라 특정 분야에 관심이 있는 사용자의 요구를 만족시킬 수 있는 임베디드시스템 개발에 대한 관심이 증가하고 있다[2]. 이러한 예로 여러 가지 악기를 이용한 연주, 편곡 혹은 작곡에 관심이 있는 사람들에게 컴퓨터 음악을 할 수 있는 환경을 제공하는 뮤직 임베디드시스템이 있다. 뮤직 시스템을 이용하면 오선지를 이용한 수작업 대신에 미디(MIDI: Musical Instrument Digital Interface) 케이블로 연결된 전자악기들을 한 파트씩 연주 및 녹음하고 음정, 속도, 피치 등을 정밀하게 편집하여 원하는 형태의 음악을 창작할 수 있으며, 창작된 여러 개의 미디파일을 동시에 재생하면 합주가 된다. 또한 재생을 하면서 미디 케이블로 연결한 악기를 연주하면 실시간 연주와 창작 연주의 합주도 가능하다. 뮤직 시스템은 악기로부터 입력되거나 미디파일로부터 재생되는 실시간 특성을 갖는 미디 데이터들을 처리할 수 있는 소프트웨어를 필요로 한다. 이 논문에서는 뮤직 소프트웨어를 설계하는데 있어서 실시간 시스템 설계 개념을 도입하여 시스템을 분석 및 설계하고, 설계된 소프트웨어 아키텍처에 대하여 성능 분석을 제시한다.

지금까지 다수의 뮤직 소프트웨어가 설계되었으나 [1, 3, 4], 이러한 설계 접근 방식들은 대부분 다음과 같은 여러 가지 문제점을 가지고 있다: (1) 설계 시에 실시간 멀티태스킹 개념을 고려하지 않았기 때문에 그 구조가 복잡하다; (2) 다른 컨커런트 모듈들과 자원의 공유에 대한 효율적인 메커니

※ 본 연구는 한국과학재단 특정기초연구(R01-2005-000-10671-0287)지원으로 수행되었음
† 준 회 원 : 울산대학교 대학원 컴퓨터공학 석사과정
†† 정 회 원 : 울산대학교 컴퓨터정보통신공학부 부교수(교신저자)
논문접수 : 2007년 11월 5일, 심사완료 : 2008년 2월 22일

즘을 제공하지 못한다; (3) 새로운 악기 모듈을 추가하기가 어려울 뿐만 아니라 추가된 모듈을 포함하는 경우 성능 분석이 어렵기 때문에 확장성이 떨어진다; (4) 프로세서 파워와 메모리 용량과 같은 자원 제한을 고려하지 않아 임베디드시스템에 적용하기 어렵다; 그리고 (5) 특정 부분의 지연 개선 문제에 중점을 두고 있으며 종합적으로 QoS를 보장하기 위한 분석 모델을 제공하지 못하고 있다.

이러한 문제를 종합적으로 해결하기 위하여 RTSAD (Real-Time Structured Analysis and Design)[7, 9]를 사용하여 뮤직 시스템을 구조를 분석하고 DARTS (Design Approach for Real-Time Systems) [5, 6]에서 제시하는 태스크 및 모듈 구조화 기준을 적용하여 실시간 뮤직 소프트웨어를 설계한다. RTSAD는 광범위하게 사용되고 있는 방식일 뿐만 아니라, 디지털 시스템의 작동 시퀀스를 이해하는데 중요한 유한 상태 천이 (FST: Finite State Transition) 다이어그램의 사용을 강조하며, 컨트롤 및 시퀀싱을 제공하기 위하여 FST 다이어그램과 DFD (Data Flow or Control Flow) 다이어그램을 통합하는 방법을 제시한다. 반면에 DARTS는 컨커런트 시스템의 설계 방법을 체계적으로 제시하고 있으며, 태스크 구조화 방법과 태스크들 사이의 인터페이스를 정의하는 방식을 강조한다 [6]. 특히, RTSAD에서 사용하는 순차적인 설계 방식 (Sequential Design Method)에서의 분석 단계와 설계 단계 사이에 태스크 구조화를 위한 절차 혹은 기준을 도입함으로써 컨커런트 설계 방식 (Concurrent Design Method)으로 어떻게 확장될 수 있는지를 보여준다.

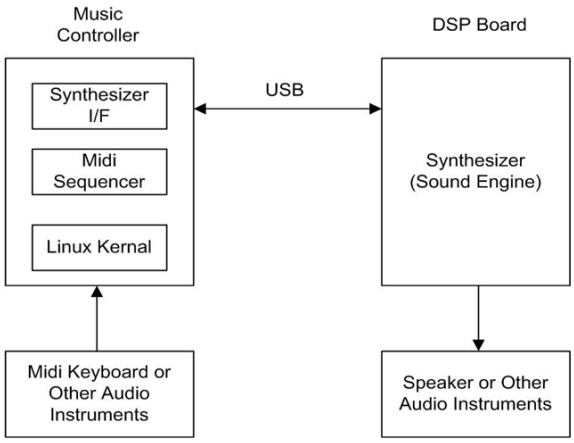
본 논문에서는 RTSAD를 사용하여 뮤직시스템을 분석하고, 분석 사양을 바탕으로 DARTS를 적용하여 복수의 입출력 스트림, 시퀀싱, 컨트롤 등을 효과적으로 다룰 수 있는 컨커런트 태스킹 소프트웨어 아키텍처를 도출한다. 또한, 각 악기의 연주 혹은 미디 파일로부터 재생되는 음악의 질 (QoS)를 보장하기 위하여 정의된 컨커런트 태스크의 실행 시간 제한 요건을 모델링 하고, 실시간 시스템의 태스크 스케줄링 분석 툴로 사용되는 RMA (Rate Monotonic Analysis) [8]를 이용하여 시스템 성능 분석 방법을 제시한다.

제 2장에서는 시스템 모델과 DARTS의 개요를 기술하고 3장에서는 RTSA 방법에서 제안하는 시스템 분석 방법에 따라서 뮤직 임베디드시스템을 분석한다. 4장에서는 컨커런트 태스크의 정의, 태스크들 사이의 인터페이스 정의, 그리고 태스크 설계를 기술한다. 5장에서는 태스크들이 컨커런트하게 실행되는 경우에 모든 태스크가 시간제약을 만족할 수 있는가에 대하여 성능 분석을 한다. 6장에서는 태스크 아키텍처의 설계내용에 대한 평가, 7장에서는 구현 내용에 대한 소개, 그리고 8장에서 결론을 맺는다.

2. 배 경

2.1 시스템 구성

(그림 1)에 도시한 뮤직 시스템은 크게 두 개의 모듈 즉,



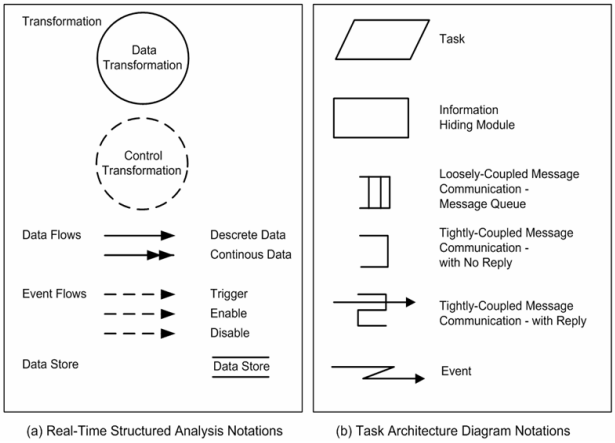
(그림 1) 뮤직 임베디드시스템의 구성

뮤직 컨트롤러에 속하는 뮤직 미디 시퀀스 모듈과 DSP 보드에 속하는 사운드 신디사이즈 모듈로 구성된다. 미디 시퀀스 모듈은 멀티 파트 (Multi-Part) 뮤직을 생성, 편집, 저장하거나 미디 인터페이스를 통해서 악기들로부터 실시간으로 미디 데이터를 받아서 처리하고, 음을 생성하기 위하여 DSP 보드의 신디사이즈 모듈로 전송한다. 신디사이즈는 뮤직 컨트롤러로부터 미디 메시지를 받아서 실시간으로 오디오 사운드를 합성한다.

2.2 시스템 분석 및 설계 방법

RTSAD [8]의 분석 방법은 기능에 의한 반복적인 분할 (Functional Decomposition)에 기반을 두고 있다. 시스템은 변환함수 (Transformation)라고 하는 함수들로 표현되고 이들 사이의 인터페이스는 DFD의 형태로 정의한다. 그 결과물인 분석 사양을 가지고 각 함수를 태스크 혹은 모듈로 맵핑한다. 하지만, 이렇게 정의된 태스크 혹은 모듈은 하나의 큰 함수 트리 구조 형태가 됨으로 실행 컨커런시를 제공하기 어렵다. 따라서, RTSA를 통해서 도출한 분석 사양에 기반을 두고 DARTS [5]가 제시하는 “컨커런트 태스크를 정의하기 위한 태스크 구조화 기준”을 적용하여 복수의 컨커런트 태스크들을 정의하고 인터페이스 가이드라인에 따라 각 태스크 사이의 인터페이스를 정의함으로써 컨커런트 태스크 아키텍처를 도출한다. 마지막으로, 구조적 설계 방법 (Structured Design Method)을 사용하여 각 태스크에 대한 구조 차트를 그린다. 이러한 과정을 통해서 시스템은 각각이 특정 역할을 수행하는 복수의 태스크들로 구성되고 태스크들은 실시간 스케줄러를 통해서 멀티태스킹 된다.

컨커런트 태스크 사이의 인터페이스는 메시지 통신, 이벤트 동기화 또는 IHM (information Hiding Module)의 형태가 된다. 메시지 통신은 약결합 (loosely coupled) 혹은 강결합 (tightly coupled)의 형태가 되며, 이벤트 동기화는 태스크 사이에 데이터가 전달되지 않는 경우에 사용한다. 공유 데이터에 대한 액세스는 IHM의 형태로 나타내고, IHM이 복수의 태스크에 의해 액세스 되는 경우에는 데이터 액세스를 동기화 해야 한다.



(그림 2) 소프트웨어 분석 및 설계에서 사용하는 기호

시스템 분석 및 설계 과정에서 함수, DFD 및 태스크 아키텍처를 표현하기 위하여 다양한 기호들을 사용한다. (그림 2)는 분석에서 사용되는 여러 가지 기호들을 나타낸다. 실선으로 된 원은 데이터 변환함수를 나타내고, 파선 원으로 표현된 제어 변환함수는 상태 천이 다이어그램을 실행하고 관리한다. 실선 화살표는 특정 시간 간격을 입력되는 데이터를 나타내며, 이중 헤드 실선 화살표는 연속적인 흐름을 가진 데이터를 표시한다. 이벤트 플로우에는 세 가지 타입이 있다: (1) Trigger에 의해 천이되는 변환함수는 한 번만 실행되는 함수이다; (2) Enable 혹은 Disable에 의해 천이되는 변환함수는 천이된 후의 상태에서 여러 번 실행될 수 있다. 데이터 공유를 위한 데이터 저장소 (Data Store)는 두 개의 평행 실선으로 표시하고 내부에 저장소 명을 표시한다. 평행사변형은 태스크를 나타내고, 오른편 하단에는 태스크 간의 통신 방식을 나타내는 4개의 표기가 있다.

3. 실시간 시스템의 구조적 분석

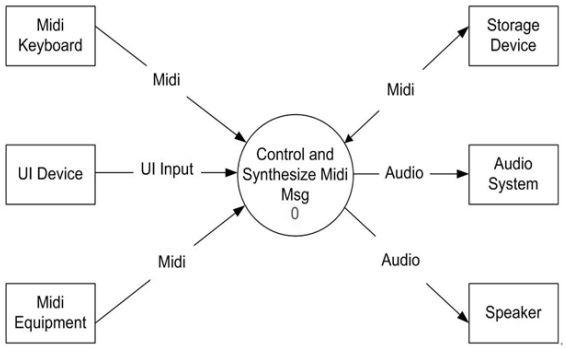
RTSA 사양은 시스템 컨텍스트 다이어그램, 계층적 DFD 다이어그램, 상태천이 다이어그램을 포함한다. 이 장에서는 이 세가지를 사용하여 구조적 분석을 수행한다.

3.1 시스템 컨텍스트 다이어그램

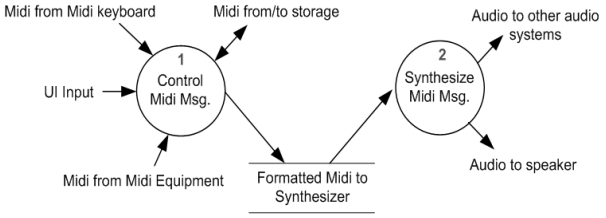
(그림 3)은 Control and Synthesize Midi Msg 모듈과 사각형으로 표시된 외부 디바이스들과의 데이터 흐름을 나타내는 컨텍스트 다이어그램이다. UI 디바이스는 터치 스크린과 마우스와 같은 장치이며, 미디 장비는 미디 키보드와 같이 미디 데이터를 생성하는 악기이다.

3.2 상위 시스템의 분할

“Control and Synthesize Midi Msg” 시스템은 (그림 4)와 같이 기능적 독립성을 가진 두 개의 하위시스템인 Control Midi Msg와 Synthesize Midi Msg으로 분할될 수 있다. 하위시스템은 독립적으로 개발되고USB와 같은 통신



(그림 3) 뮤직 임베디드시스템의 컨텍스트 다이어그램

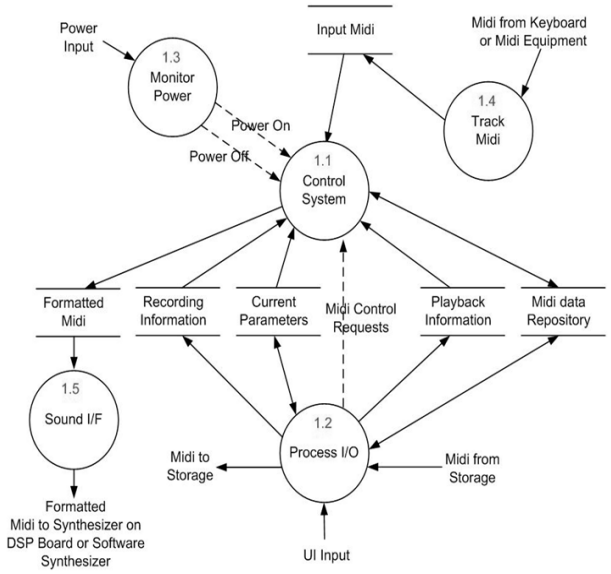


(그림 4) Control and Synthesize Midi Msg DFD

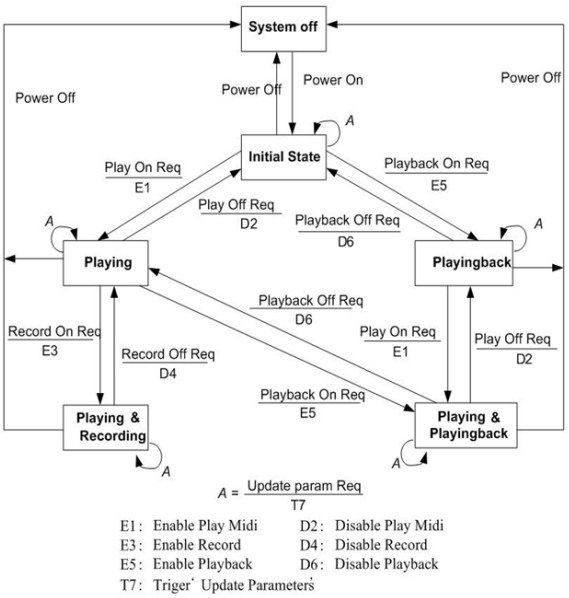
인터페이스를 사용하여 통합된다. 이 논문에서는 Control Midi Msg 하위 시스템만 기술하며 Synthesizer Midi 하위 시스템은 DSP 보드가 되거나 소프트웨어 신디사이즈 모듈이 된다.

3.3 Control Midi Msg 하위시스템

Control Midi Msg 하위시스템은 (그림 5)에 나타난 것처럼 5개의 데이터 변환함수로 세분화될 수 있다. 파워 버튼을 누르면 모든 태스크들이 초기화된다. Monitor Power 데이터 변환함수는 파워 시그널을 감지하고 파워 ON 혹은 OFF 시그널을 Control System 데이터 변환함수로 보낸다. Track Midi는 미디 키보드 혹은 미디 악기로부터 Midi 데이터를 읽어 들여 Input Midi 데이터 저장소에 저장하고, Control System 데이터 변환함수는 저장된 데이터를 필요할 때 액세스 한다. Control System 데이터 변환함수는 사용자와의 상호작용을 처리하는 Process IO 변환함수로부터 Midi Control Requests 이벤트를 받고 요구 사항에 따라서 적합한 함수를 수행한다. 사용자 인터페이스를 담당하는 Process IO 변환함수에 의해 변경되는 악기 유형, 피치, 사운드 효과 등을 나타내는 Current Parameters 와 Input Midi 혹은 Midi Data Repository에 저장된 미디 데이터는 Control System 변환함수에 의해 신디사이즈에서 요구되는 미디 포맷으로 변환되어 Formatted Midi Data Repository에 저장된다. Sound I/F 함수는 저장된 데이터를 주기적으로 읽어서 DSP신디사이즈 혹은 소프트웨어 신디사이즈로 전송한다. Playback Information은 Midi Data Repository 데이터 버퍼에 있는 미디 파일 중에서 재생될 영역의 트랙 번호 및 트랙 범위에 대한 정보를 가지고 있으며, Recording Information 데이터 저장소는 특정 미디 디바이스에 대응하는 입력 채널



(그림 5) Control Midi Msg DFD



(그림 6) Control State Transition(CST) 다이어그램

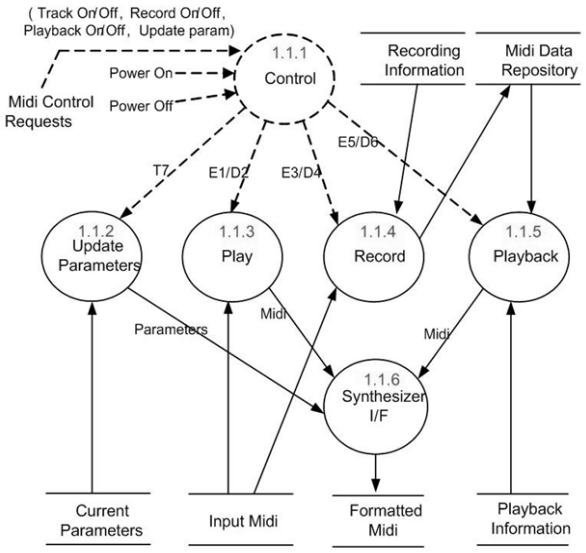
번호를 가지고 있다.

3.4 Control System 변환함수

(그림 7)의 DFD는 (그림 5)의 Control System에 대한 것으로, Control 제어 변환함수는 Process IO 혹은 Monitor Power 데이터 변환함수들로부터 Midi Control Requests 혹은 Power On/Off 이벤트를 받으면, 현재 상태와 수신 이벤트에 따라 해당 함수를 실행한다. 따라서, Control 제어 변환함수는 (그림 6)에 있는 CST 다이어그램의 실행을 통하여 시스템을 제어한다.

CST 다이어그램은 여섯 개의 상태를 가지며 상태 천이와 각 상태 천이와 관련된 행위를 나타낸다. Power On 이벤트를 수신할 때 Initial State가 되며 모든 태스크 및 파라미터가 초기화 된다. Play On Req (Playback On Req) 이벤트를 수신할 때 시스템은 Playing (Playingback) 상태에 들어간다. Playing 상태에서는 실시간으로 미디 키보드/미디 악기로부터 읽은 미디 데이터를 재생하고, Playingback 상태에서는 Midi Data Repository로부터 미디 데이터를 재생한다. Playing 상태에서 Playback On Req 이벤트를 받거나 Playingback 상태에서 Play On Req를 받으면 Playing & Playingback 상태가 되고, 악기로부터 받은 미디 데이터와 Midi Data Repository로부터 읽은 미디 데이터를 동시에 재생한다. 이런 식으로 복수의 악기들로부터 읽은 미디 데이터를 재생하고 동시에 복수의 Midi Data Repository로부터 미디 데이터를 읽어 음을 재생함으로써 여러 악기의 합주가 가능하다.

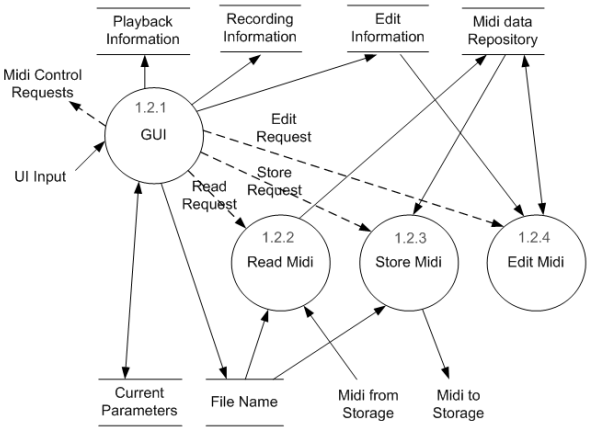
Synthesizer I/F는 Play 및 Playback 변환함수로부터 미디 데이터, Update Parameters 변환함수로부터 파라미터 값을 수신하여 신디사이즈가 요구하는 포맷으로 변경하여 Formatted Midi 저장소에 저장한다.



(그림 7) Control System DFD

3.5 Process IO 변환함수

(그림 5)의 Process IO 데이터 변환은 (그림 8)과 같이 사용자 인터페이스, 파일 인터페이스, 그리고 미디 파일 편집과 관련된 여러 가지 변환함수로 세분화 된다. GUI 데이터 변환함수는 UI Input을 취하고 그것을 대응하는 내부 이벤트로 변환하여 해당 변환함수로 전달한다. 미디 제어와 관련된 Midi Control Requests를 (그림 7)의 Control 제어 변환함수로 보내고, Edit/Read/Store관련 입력 이벤트를 대응하는 변환함수로 보낸다. 이 DFD에서 새로 소개된 Edit Information 저장소는 Midi Data Repository에 있는 미디 데이터 중에서 편집할 미디 데이터의 범위, 삽입 혹은 삭제와 같은 편집관련 세부 정보를 가진다.



(그림 8) Process IO DFD

4. 태스크 구조화

4.1 컨커런트 태스크

소프트웨어 분석 스펙에 기반을 두고 각 데이터 및 제어 변환함수를 컨커런트 태스크로 맵핑하기 위하여 다음과 같은 3가지 태스크 구조화 기준을 사용한다. (1) DFD에서 외부 디바이스 인터페이스를 담당하는 각 I/O 변환함수는 각 디바이스의 데이터 처리 특성이 다르기 때문에 독자적인 비동기 I/O 태스크, 주기적인 I/O 태스크 또는 자원 모니터 태스크로 맵핑된다; (2) DFD의 내부 변환함수는 컨트롤 태스크, 주기적 태스크 또는 비동기적 태스크로 맵핑된다; 그리고 (3) DFD의 내부 변환함수는 순차적 (Sequential) 응집력, 시간적 (Temporal) 응집력, 기능적 (Functional) 응집력의 정도에 따라 다른 변환함수와 통합될 수 있다.

동일한 주기를 가진 이벤트들의 발생에 의해서 실행되는 변환함수들은 대부분의 경우에 시간적 응집력에 의해 결합될 수 있다. 비동기적 혹은 주기적인 이벤트의 발생에 의해 실행되는 변환함수 다음에 반드시 어떤 변환함수가 실행되어야 한다면 두 함수는 순차적 응집력에 의해 결합될 수 있다. 두 개 이상의 변환함수가 기능적으로 밀접하여 두 변환함수 사이에 데이터 트래픽이 많은 경우 혹은 여러 개의 변환함수가 동일한 공유자원 혹은 동일한 I/O 디바이스를 액세스하는 경우에 결합될 수 있다.

기준 (1), (2)에 따라 DFD는 13개의 태스크 (1.1.1, 1.1.2, 1.1.3, 1.1.4, 1.1.5, 1.1.6, 1.2.1, 1.2.2, 1.2.3, 1.2.4, 1.3, 1.4, 1.5)로 맵핑될 수 있지만, 태스크의 수가 증가하면 시스템 오버헤드가 증가하기 때문에 표 1은 위의 세 번째 기준을 추가로 적용하여 정의한 8개의 태스크들을 요약한다. 단, Power On/Off를 모니터링하는 Monitor Power 변환함수 (그림 5)의 1.3는 시스템이 켜질 때 시스템은 즉시 Initial 상태로 들어가기 때문에 제외한다. (그림 7)의 Update Parameters 변환함수 (1.1.2), Play 변환함수 (1.1.3), 그리고 Playback 변환함수 (1.1.5)는 실행될 때 반드시 미디 포맷 변환을 수행하는 Synthesizer I/F 데이터 변환함수 (1.1.6)를 호출하기 때문에 변환을 담당하는 Synthesizer I/F 데이터 변환 함수는 각 선행하는 변환함수에 포함시킨다.

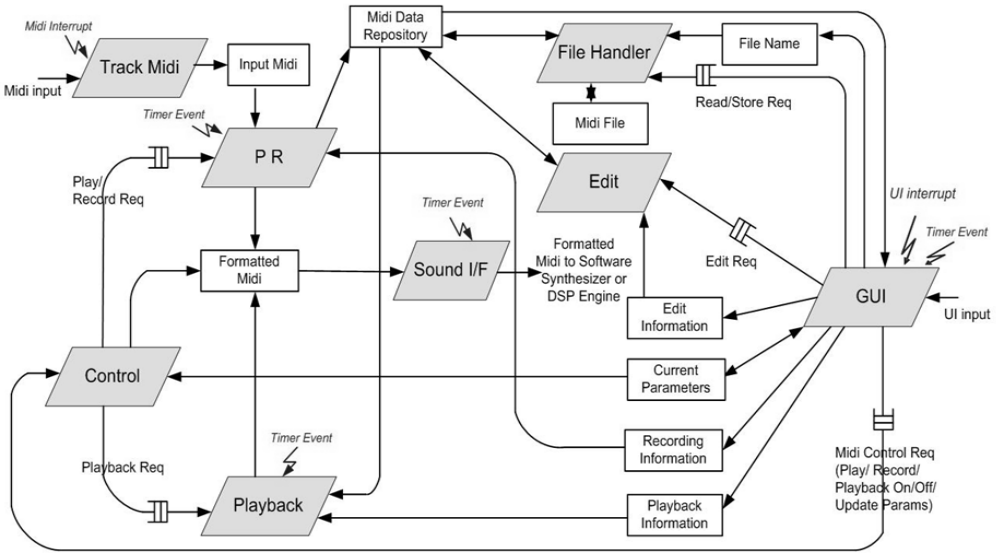
〈표 1〉 태스크 구조화 기준을 적용하여 정의한 태스크

태스크	대응하는 변환함수 및 적용 기준
Track Midi	Track Midi 데이터 변환함수(그림 5의 1.4)는 미디 디바이스로부터 미디 데이터를 트랙킹하여 Input Midi 버퍼에 저장한다. 이 I/O 변환함수는 비동기 주기적인 Track Midi 태스크로 맵핑된다.
Sound I/F	Sound I/F 데이터 변환함수(그림 5의 1.5)는 사운드 신디사이즈를 위해 Formatted Midi 데이터 저장소에 저장된 미디 데이터를 사운드 엔진으로 보내는 주기적인 Sound I/F 태스크로 맵핑된다.
Control	Control 제어 변환함수(그림 7의 1.1.1)는 (그림 6)의 상태천이 다이어그램을 실행하는 컨트롤 태스크로 맵핑된다. 또한, Update Params 이벤트가 들어오면 최우선으로 파라미터 변경을 하여야 함으로 Update Parameters 데이터 변환함수(그림 7의 1.1.2)는 기능적인 응집력에 의해서 Control 태스크로 통합된다.
PR	Play 데이터 변환함수(1.1.3)와 Record 데이터 변환함수(1.1.4)는 Input Midi에서 미디 데이터를 읽어 포맷을 변경하여 Formatted Midi 데이터 저장소에 넣고, 동시에 Midi Data Repository에 레코딩하는 것을 연속적으로 수행해야 한다. 따라서, 시간적, 연속적인 응집력이 강하므로 결합되어 PR 태스크가 된다.
Playback	Playback 데이터 변환함수는 Midi Data Repository로부터 미디 데이터를 읽고, 읽은 미디 데이터를 Synthesizer I/F 라이브러리를 이용하여 변환한 후에 Formatted Midi 버퍼에 저장하는 주기적인 Playback 태스크로 맵핑된다.
GUI	GUI 변환함수는 외부 입력 혹은 내부 이벤트 발생과 같은 내외부 인터럽트에 의해서 시동되는 비동기적 GUI 태스크로 맵핑 된다. 인터럽트는 터치스크린이나 재생시간 진행을 나타내는 타이머에 의해서 발생할 수 있다.
File Handler	기능적 응집력이 강한 Read Midi 데이터 변환함수(1.2.2)와 Store Midi 데이터 변환함수(1.2.3)는 둘 다 보조 기억장치의 미디 파일을 취급하기 때문에 File Handler 태스크로 맵핑된다.
Edit Midi	Edit Midi 데이터 변환함수 (1.2.4)는 Edit Information 데이터 저장소에 있는 정보를 이용하여 Midi Data Repository에 있는 미디 데이터를 편집하는 Edit 태스크로 맵핑된다.

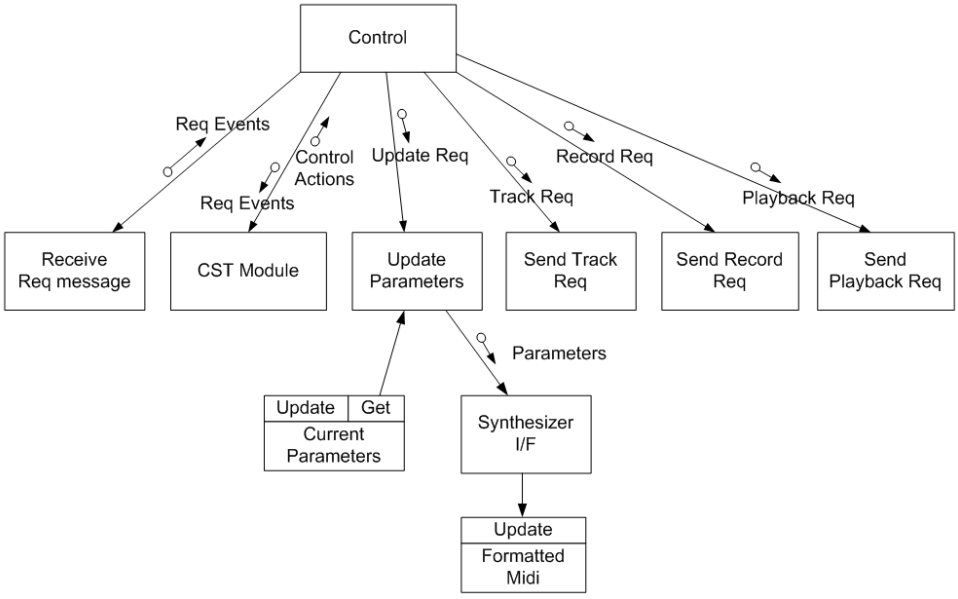
4.2 컨커런트 태스크 아키텍처

<표 1>에 요약한 태스크들을 기반으로 뮤직 임베디드 소프트웨어에서 필요한 태스크들과 그 태스크들 사이의 인터페이스를 (그림 9)에 도시하였다. 이 태스크 아키텍처는 위에서 분석한 DFD 플로우에 나타난 데이터 변환함수 및 컨트롤 변환함수들을 DARTS 태스크 구조화 기준을 적용하여 도출한 것이다.

GUI는 비동기적인 태스크로서 사용자 입력 혹은 음악 연주의 끝을 알려주는 타이머와 같은 인터럽트에 의해 실행된다. PR 태스크와 Playback 태스크는 Recording Information에 따라서 각각 Input Midi 데이터 버퍼와 Midi Data



(그림 9) 태스크 아키텍처 다이어그램



(그림 10) Control 태스크에 대한 구조 차트

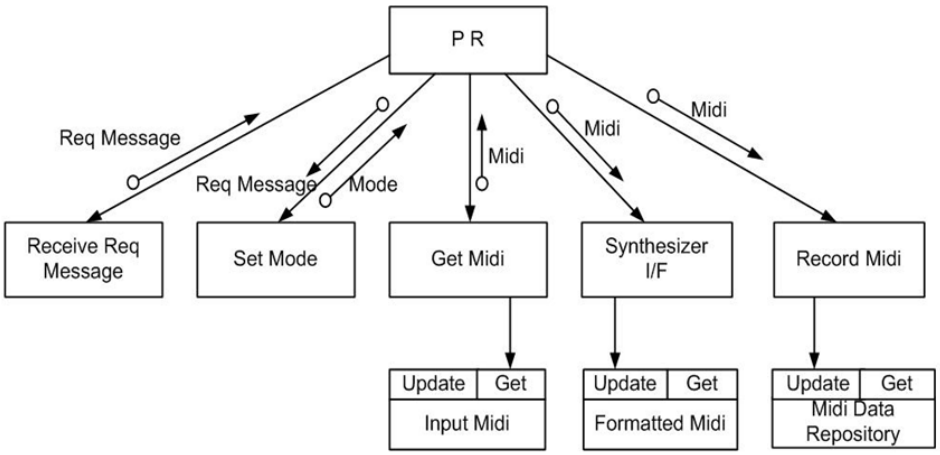
Repository 데이터 버퍼로부터 주기적으로 Midi 데이터를 읽고, 읽은 미디 데이터를 소프트웨어 신디사이즈나 혹은 DSP 사운드 엔진이 인식할 수 있는 형태로 포맷을 변경하여 Formatted Midi에 저장한다. Control 태스크는 상태천이도를 실행한다.

4.3 태스크 구조 설계

구조 차트는 태스크를 모듈들로 세분화하여 연속적인 실행 형태로 나타낸다. 이 논문에서는 2개의 태스크에 대한 구조 차트 만을 제시한다.

(그림 10)은 Control 태스크의 구조 차트이다. Control 태스크는 새로운 이벤트를 수신하면 Req Events를 CST Module에 전달하고 Control Actions을 얻으며, 그 결과에 따라 Control 태스크는 대응하는 하부 모듈을 호출한다. 예를 들면, Control 태스크가 현재 Play 상태에서 GUI 태스크로부터 Record Req 이벤트를 수신하면, 수신한 이벤트를 CST Module에 보내고, (그림 6)의 상태천이 다이어그램에 따라 E3 (Enable Record) Action을 리턴 받게 된다. Control 태스크는 Enable Record를 수행하기 위해서 PR 태스크에게 Record Req 이벤트를 전송하는 모듈인 Send Record Req를 호출하게 된다.

(그림 11)의 PR 태스크는 Receive Req Message 모듈을 호출하여 Control 태스크로부터 Req Message를 받고, 수신한 Req Message를 Set Mode 모듈에게 보낸다. Set Mode



(그림 11) PR 태스크의 구조 차트

모듈은 Req Message를 분석하여 모드를 셋하고 새로운 모드를 리턴한다. 모드는 다음 3가지 모드 즉, Play, Play & Record, Record 중의 하나가 된다. 만약 이전에 PR 태스크가 작동하고 있지 않은 상태에서 Play Req를 수신한 경우에 새로운 모드는 Play가 되고, PR 모듈은 주기적으로 Input Midi 버퍼에서 미디 데이터를 읽고, 읽은 데이터를 Synthesizer I/F를 통해 포맷 변환한 후에 Formatted Midi에 저장한다. 이러한 반복적인 작업은 Control 태스크가 Disable Play Midi (그림 7의 D2)를 보낼 때까지 계속된다. Formatted Midi 및 Midi Repository는 공유 저장소이므로 IHM로 구현되고 액세스 동기화를 위해서 Update 및 Get 액세스 함수는 세마포어를 사용한다.

〈표 2〉 태스크 특성

Task	CPU time (msec)			
	C_i	Periodic	Aperiodic	Event Sequence Task
*GUI Interrupt (C_1)	1			
*GUI (C_2)	2		2	
PR (C_3)	3	$4 = C_3 + 2C_x$		
Playback (C_4)	2	$3 = C_4 + 2C_x$		
*Control (C_6)	2		2	
File Handler (C_7)	6		6	
Edit Midi (C_8)	6		6	
Track Midi (C_9)	1		1	
Sound I/F(C_{10})	1	$2 = C_{10} + 2C_x$		
Context Switching Overhead (C_x)	0.5			
Message Communication Overhead (C_m)	1			
*Event Sequence (C_e)				$8.5 = C_1 + C_2 + C_6 + 3C_x + 2C_m$

* GUI Interrupt, GUI, and Control 태스크들은 하나의 "Event Sequence 태스크"로 정의되고, 실행 시간 C_e를 가진 주기적인 태스크로 취급된다. C_i 값은 측정값으로부터 얻은 최대 실행 시간이다.

5. 성능 분석

모든 태스크들이 실행 시간의 제약을 만족할 수 있는지를 테스트하기 위해서 RMA를 사용하여 성능을 분석한다. 실시간으로 입력되는 미디 데이터들을 트랙킹하는 Track Midi 태스크는 음의 질적인 면을 고려하여 가장 높은 우선순위를 할당한다. PR 태스크와 Playback 태스크는 각각 서로 다른 음원 즉, Input Midi 및 Midi Data Repository로부터 20ms 마다 미디 데이터를 읽어서 신디사이즈가 요구하는 포맷으로 변환하고, Formatted Midi 데이터 버퍼에 저장한다. Sound I/F 태스크는 10ms 마다 주기적으로 Formatted Midi에 있는 데이터를 DSP사운드 모듈로 보낸다. 이들 세 개의 태스크는 비교적 엄격한 시간 제한을 갖는 주기적인 태스크이다. Control 태스크는 네 번째 높은 우선순위를 할당하며 하나의 입력 이벤트를 처리하기 위하여 2ms의 시간이 요구된다. GUI 태스크는 Control 태스크가 쉬고 있는 경우에도 입력 이벤트를 처리해야 함으로 다섯 번째 우선순위를 할당한다. File Handler 태스크와 Edit Midi 태스크는 가장 낮은 우선 순위를 할당하여 CPU가 한가한 시간에 처리하도록 한다. 모든 실행 시간 마이크로프로세서의 파워에 따라 변한다. 표 2는 태스크 특성을 요약한다.

주기적인 실시간 태스크들은 실행하고 있는 동안에도 다음에 일련의 이벤트들은 우선적으로 처리해야 한다. 실행해야 할 이벤트 시퀀스는 다음과 같다:

- GUI 인터럽트가 발생한다 (C_1)
- GUI가 UI 디바이스로부터 입력을 읽는다 (C_2)
- GUI 태스크가 Control Req 메시지를 Control 태스크에 보낸다 (C_m)
- Control 태스크가 메시지를 받고, 상태천이도를 실행하고, 입력 이벤트에 따라 상태를 바꾼다 (C_6)
- Control 태스크가 외부로 나가는 화살표에 의해 연결된 태스크들 중의 하나에 대응하는 요청 메시지를 보낸다 (C_m)

$C_e = C_1 + C_2 + C_6 + 3C_x + 2C_m = 1 + 2 + 2 + 3 * 0.5 + 2 * 1 = 8.5$ms. 이 이벤트 시퀀스를 ES라고 하면, ES는 100ms의 최대 응답 시간을 가지고 실행하는 하나의 주기적인 태스크로 모델 될 수 있다. 즉, ES 태스크는 이벤트 시퀀스로써 비 주기적으로 발생하지만 최악의 경우에 C_e 만큼 주기적으로 CPU를 사용한다고 가정하고 스케줄 가능성을 검증한다.

최악의 경우 PR 태스크의 실행시간은 다음과 같은 실행 시나리오에서 발생한다:

- PR 태스크가 메시지 큐를 체크하고 Play 모드에서 Play & Record 모드로 모드를 변경한다.
- PR 태스크가 미디 데이터를 읽고 레코딩을 위해서 읽은 미디 데이터를 Midi Data Repository에 쓴다.
- PR 태스크가 미디 데이터를 Synthesizer I/F 함수를 호출하여 사운드 신디사이즈에 맞게 포맷을 바꾼다.
- PR 태스크가 포맷 변경된 미디를 Formatted Midi 데이터 버퍼에 쓴다.

$C_{TR} = C_3 + 2C_x = 3 + 2 * 0.5 = 4$ms. 이 태스크는 20ms 마다 주기적으로 호출된다.

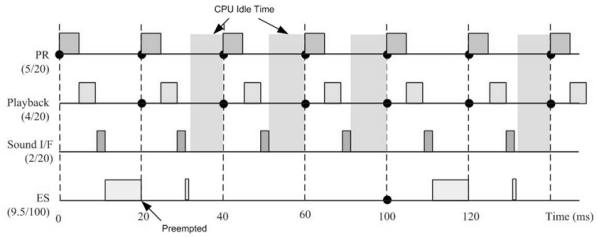
최악의 경우 Playback 태스크의 실행시간은 다음과 같은 실행 시나리오에서 발생한다:

- Playback 태스크는 자신의 큐를 체크하여 Playback Off 혹은 Playback On의 요청이 있는지를 알아본다.
- Playback 태스크는 Midi Data Repository로부터 미디 데이터를 읽고 Synthesizer I/F 라이브러리 함수를 호출하여 사운드 신디사이즈 포맷으로 변환한다.
- Playback 태스크는 포맷 변환된 미디데이터를 Formatted Midi 데이터 버퍼에 쓴다.

$C_{Playback} = C_4 + 2C_x = 2 + 2*0.5 = 3$ms. 이 태스크는 20ms 마다 주기적으로 호출된다.

〈표 3〉 태스크 파라미터 값

Task Name	Type of Task	Execution Time C_i	Period T_i	Priority
*PR	Periodic	5 = $C_{PR} + C_{TM}$	20ms	2
*Playback	Periodic	4 = $C_{Playback} + C_{TM}$	20ms	2
*Sound I/F	Periodic	2	20ms	2
*ES (GUI, Control)	Modeled Periodic	9.5 = $C_e + C_{TM}$	100ms	3
Track Midi	Aperiodic	1		1
File Handler	Aperiodic	-		4
Edit Midi	Aperiodic	-		4



(그림 12) 태스크 실행 타이밍 차트

<표 3>에서 * 마크를 가진 태스크들은 높은 우선순위를 가진 비동기적 태스크인 Track Midi 태스크로부터 실행 중에 한 번의 선취를 당할 수 있기 때문에 이러한 것을 감안하여 C_{TM}을 더하였다. 주기적인 태스크 셋 Γ = {PR, Playback, Sound I/F, ES}의 CPU 이용률, U_Γ = 0.645 (= 5/20 + 4/20 + 2/20 + 9.5/100)가 된다. RMA [7]에 의하면, 태스크 셋은 다음 조건이 만족되는 경우에 모든 태스크들이 스케줄 가능한 것으로 판단한다:

$$U(n) = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

만약에 n이 4가 되면, $U(4) = 4(2^{1/4} - 1) = 0.756$. 따라서, $U\Gamma \leq U(4)$이기 때문에 주어진 주기적인 태스크 셋은 스케줄이 가능한 것으로 판단된다.

시스템 타이밍 다이어그램은 (그림 12)에 있으며, 각 태스크는 주기의 끝이 되는 데드라인에 대하여 성공적으로 실행됨을 알 수 있다. CPU가 쉬고 있는 시간 (Idle Slot)동안에 시간적으로 덜 중요한 File Handler 태스크와 Edit Midi 태스크가 실행된다.

6. 설계 평가

제안한 뮤직 임베디드 소프트웨어 아키텍처의 모든 태스크들은 실시간 스케줄러에 의해 멀티태스킹을 할 수 있도록 설계되었다. 컨커런트 태스크들끼리 공유되는 자원은 IHM로 구현되고 액세스 동기화 메커니즘을 제공함으로써 동기화 문제를 해결할 수 있다. RMA 성능 분석 모델을 사용함으로써 시스템 성능을 체계적으로 분석할 수 있으며 QoS 문제를 미리 진단할 수 있다. 새로운 악기를 추가하는 경우에 Track Midi 태스크를 수정하고 QoS 만족할 수 있는지에 대하여 성능 분석을 함으로써 추가 가능성을 평가할 수 있으며, 다른 태스크의 구조 변경이 필요 없다. 또한, 각 태스크가 사용하는 프로세서 이용률과 메모리 사용 내역을 분석하고 문제가 되는 태스크 만을 최적화 함으로써 성능을 개선을 할 수 있다. 데이터 처리 지연으로 인하여 음질이 떨어지는 경우에는 중요 실행 경로 (Critical Path) 상에 있는 태스크들의 우선 순위, 실행 주기, 실행시간 등을 변경함으로써 해결책을 찾을 수 있다.

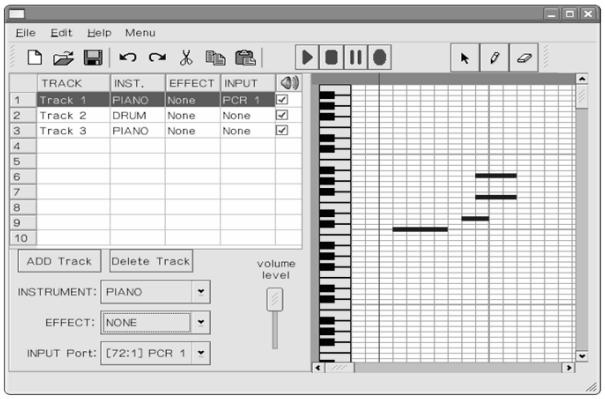
7. 구 현

7.1 구현 환경

설계한 뮤직 소프트웨어를 Linux 2.6을 탑재한 64MB 플래시 메모리를 가진 Xhyper272에서 구현하였다. Linux 커널, 폰트, 그리고 ALSA (Advanced Linux Sound Architecture)와 같은 시스템 파일들을 설치한 후에 남은 30.8MB 중에 5MB를 뮤직 임베디드 소프트웨어의 실행 코드를 저장하는데 사용하였다.

7.2 태스크 구현

Track Midi 태스크는 리눅스 시스템의 ALSA 드라이브에 대응한다. PR 태스크는 미디 키보드로부터 미디를 받아들이기 위해서 ALSA 시퀀스 모듈을 이용한다. GUI 태스크를 구현하기 위해서 리눅스 기반의 임베디드 소프트웨어 개발을 위한 C++ GUI 툴인 Qt/Embedded를 사용하였다. (그림 13)은 GUI 화면을 보여준다. 태스크 사이의 통신이 상당한 시스템 오버헤드를 발생시킨다는 것을 고려하여, 태스크 간의 통신에서 시스템 콜을 사용하는 대신에 미리 정의된 버퍼를 사용하여 통신을 하도록 하였다.



(그림 13) 뮤직 임베디드시스템의 GUI

7.3 GUI 작동 방식

GUI 태스크는 그래픽으로 데이터를 표현하며 사용자가 데이터를 조작하고 함수를 선택하기 위해서 대응하는 그림을 클릭함으로써 뮤직 시스템을 작동할 수 있도록 편리한 사용자 인터페이스를 제공한다. GUI는 미디 데이터 편집, 신디사이즈 파라미터 변경, 재생, 레코딩, 그리고 파일 관리와 같은 기능들을 제공한다.

(그림 13)은 GUI 태스크의 주 화면이다. 우측에 있는 그리드 영역을 이용하여 미디 데이터를 편집할 수 있다. 수평 및 수직선은 각각 시간 값과 음의 피치를 나타낸다. 그리드 상의 푸른 색의 수평 바 (Bar)는 미디 데이터를 나타낸다. 빨강색 수직 선은 재생되는 시간을 나타낸다. 툴 메뉴 바의 팬과 지우개를 사용하여 미디 데이터를 추가 혹은 삭제할 수 있다. 특정 악기에 해당하는 트랙의 사운드를 듣기 위해서 좌측의 트랙 정보 테이블에 있는 대응하는 체크 박스를 클릭하면 된다. 메뉴 바의 빨강색 아이콘을 클릭하면 레코딩을 시작하고 레코딩을 하는 동안에 기록되는 미디 데이터는 오른쪽 그리드 영역에 나타난다. 좌측 하단의 스크롤 버튼을 사용하여 악기, 효과, 그리고 볼륨 레벨 등에 대한 신디사이즈 파라미터 값을 변경할 수 있다. 악기 및 효과 파라미터 값들은 각 출력 채널에 대하여 서로 다르게 조정될 수 있지만, 볼륨 레벨은 모두에게 공통으로 적용된다.

8. 결 론

기존의 컴퓨터 음악 소프트웨어들은 멀티미디어 데이터를 처리함에도 불구하고 실시간 소프트웨어 설계 개념을 적용하지 않고 하나의 응용 소프트웨어처럼 설계되어 확장성이 낮고 성능 분석이 어렵다. 이러한 문제를 해결하기 위하여 RTSA및 DARTS를 이용하여 뮤직 임베디드 시스템을 분석하고 태스크 아키텍처를 설계 및 구현하였다. 제안한 소프트웨어 아키텍처를 사용하면 각 태스크의 실행 주기 및 시간을 이용하여 시스템이 원하는 QoS를 만족할 수 있는지를 분석할 수 있으며, 쉽게 새로운 악기를 추가할 수 있는 장점이 있다.

참 고 문 헌

[1] Brandt, E. and Dannenberg, R. B., “Low-latency music software using off-the-shelf operating systems,” In Proc. 1998 Intl. Computer Music Conf. (ICMC-98), San Francisco, pages 137-141, 1998.

[2] Buttazzo, G., “Research trends in real-time computing for embedded systems,” ACM SIGBED Review, vol.3, issue 3, (July 2006) Pages 1-10

[3] Chaudhary, A., Freed, A. and Wright, M. “An open architecture for real-time music software,” In Proceedings of the 2000 International Computer Music Conference, (Berlin, 2000). ICMA, San Francisco, 2000, 492-495

[4] Dannenberg, R. B. "Aura II: making real-time systems safe for music," In Proceedings of the International Conference on New Interfaces for Musical Expression. Hammamatsu, Japan, 2004.

[5] Gomaa, H., “Using the DARTS software design method for real-time systems,” Proceedings of the Twelfth Structured Metods Conference, Chicago, Aug., 1987.

[6] Gomma, H., “Strucruring Criteria for Real-Time System Design,” Proceedings of the Eleventh International Conference on Software Engineering, May, 1989.

[7] Hatley, D. and Pirbhai, I., Strategies for Real-Time System Specification, New York: Dorset House, 1988.

[8] Liu, C. L. and Layland, J. W., “Scheduling algorithm for multiprogramming in a hard real-time environment,” Journal of the ACM, vol.20, Jan., 1973.

[9] Ward, P., and S. Mellor, “Structured development for real-time systems,” Three Volumes, Englewood Cliffs, N. J., Prentice Hall, 1985

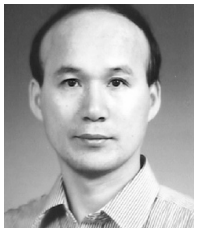
최 성 민

e-mail : galfosman@hanmail.net

2006년 울산대학교 컴퓨터정보통신공학부 (공학사)

2006년~현 재 울산대학교 대학원 컴퓨터공학 석사과정

관심분야 : 임베디드시스템, 실시간 컴퓨팅



오 훈

e-mail : hoonoh@ulsan.ac.kr

1981년 성균관대학교 전자공학(학사)

1992년 텍사스A&M대학교 전산학(석사)

1995년 텍사스A&M대학교 전산학(박사)

1996년 삼성전자 중앙연구소 수석연구원

2005년 울산대학교 컴퓨터정보공학부 조교수

2008년~현 재 울산대학교 컴퓨터정보통신공학부 부교수

관심분야 : 실시간 컴퓨팅, Ad Hoc 및 센서 네트워크 프로토콜, 임베디드시스템