(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2013-0055518
(43) 공개일자 2013년05월28일

(51) 국제특허분류(Int. Cl.)
G06F 11/36 (2006.01) *G06F 11/28* (2006.01)
G06F 9/44 (2006.01)
(21) 출원번호 10-2012-0121549
(22) 출원일자 2012년10월30일
심사청구일자 2012년10월30일
(30) 우선권주장
1020110121049 2011년11월18일 대한민국(KR)

(71) 출원인
고려대학교 산학협력단
서울 성북구 안암동5가 1
(72) 발명자
차성덕
서울특별시 성북구 종암동 삼성래미안2차아파트 210-1401
김희진
서울특별시 서대문구 대현동 11-1 이화여자대학교 아산공학관 321-2호
(뒷면에 계속)
(74) 대리인
특허법인 신성

전체 청구항 수 : 총 10 항

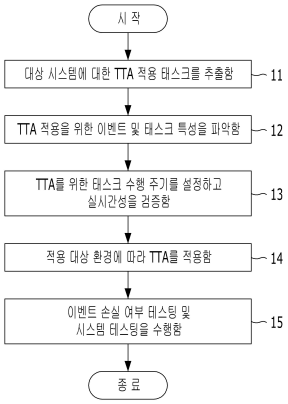
(54) 발명의 명칭 **실시간 내장형 시스템을 위한 TTA 적용 방법**

(57) 요 약

본 발명은 실시간 내장형 시스템을 위한 TTA 적용 방법에 관한 것으로, 기존 ETA 기반 실시간 소프트웨어를 TTA 기반 실시간 소프트웨어로 개선하는, 실시간 내장형 시스템을 위한 TTA 적용 방법 및 상기 방법을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체를 제공하고자 한다.

이를 위하여, 본 발명은 실시간 내장형 시스템을 위한 TTA(Time-Triggered Architecture) 적용 방법에 있어서, (a) 대상 시스템에 대한 TTA 적용 태스크를 추출하는 단계; (b) TTA 적용을 위한 이벤트 및 태스크 특성을 파악하는 단계; (c) TTA를 위한 태스크 수행 주기를 설정하고 실시간성을 검증하는 단계; (d) 적용 대상 환경에 따라 TTA를 적용하는 단계; 및 (e) 이벤트 손실 여부 테스팅 및 시스템 테스팅을 수행하는 단계를 포함한다.

대 표 도 - 도1



(72) 발명자

선경

서울특별시 노원구 월계3동 18 그랑빌아파트 116-302

안치범

서울특별시 강북구 수유1동 472-283

김영갑

경기도 과천시 별양동 주공아파트 614-303

박상수

서울특별시 서대문구 대현동 11-1 이화여자대학교 아상공학관 321-2호

정세훈

서울특별시 성북구 안암동2가 155-5 4층

최재순

경기도 고양시 일산서구 대화동 성저마을4단지아파트 406-402

이 발명을 지원한 국가연구개발사업

과제고유번호	ITAB1100110100020001000200200
부처명	한국소프트웨어진흥원
연구사업명	SW공학 요소기술 개발과 전문인력 양성사업
연구과제명	융합소프트웨어 신뢰성 기술개발 연구
주관기관	고려대학교 산학협력단
연구기간	2011.01.01 ~ 2011.12.31

특허청구의 범위

청구항 1

실시간 내장형 시스템을 위한 TTA(Time-Triggered Architecture) 적용 방법에 있어서,

(a) 대상 시스템에 대한 TTA 적용 태스크를 추출하는 단계;

(b) TTA 적용을 위한 이벤트 및 태스크 특성을 파악하는 단계;

(c) TTA를 위한 태스크 수행 주기를 설정하고 실시간성을 검증하는 단계;

(d) 적용 대상 환경에 따라 TTA를 적용하는 단계; 및

(e) 이벤트 손실 여부 테스팅 및 시스템 테스팅을 수행하는 단계

를 포함하는 TTA 적용 방법.

청구항 2

제 1 항에 있어서,

상기 (a) 단계는,

이벤트를 구분하는 과정;

태스크 진입점을 구분하는 과정;

서브 루틴을 구분하는 과정;

공유 자료구조를 구분하는 과정; 및

정의-사용(Definition-Use) 관계를 파악하는 과정

을 포함하는 TTA 적용 방법.

청구항 3

제 1 항에 있어서,

상기 (a) 단계를 "전용 소프트웨어 분석 도구를 사용하는 데이터 플로우 분석 기법"을 이용하여 수행하는, TTA 적용 방법.

청구항 4

제 1 항에 있어서,

상기 (b) 단계는,

대상 이벤트의 발생 주기를 파악하는 과정;

관련 태스크의 최악 수행 시간 및 우선순위의 실시간 특성을 파악하는 과정; 및

태스크에 시스템 자원을 분배하는 스케줄러의 특성을 파악하는 과정

을 포함하는 TTA 적용 방법.

청구항 5

제 4 항에 있어서,

상기 최악 수행 시간의 실시간 특성을 파악하는 과정은,

타이머를 선택하는 과정;

코드를 삽입하는 과정;

정보 획득 경로를 설정하는 과정;

테스트 케이스를 작성하는 과정; 및

상기 작성된 테스트 케이스를 수행하는 과정

을 포함하는 TTA 적용 방법.

청구항 6

제 5 항에 있어서,

상기 테스트 케이스를 작성하는 과정은,

각 태스크의 소스코드와 외부에서 발생하는 이벤트 특성을 분석하여 각 태스크별로 최악 수행 시간이 발생할 수 있는 테스트 케이스와, 각 이벤트별로 최대 수행 빈도 및 최소 수행 빈도가 발생할 수 있는 테스트 케이스를 작성하는, TTA 적용 방법.

청구항 7

제 4 항에 있어서,

상기 최악 수행 시간의 실시간 특성을 파악하는 과정을 "최악 수행 시간(WCET: Worst-Case Execution Time) 분석 도구를 사용하는 기법"을 이용하여 수행하는, TTA 적용 방법.

청구항 8

제 1 항에 있어서,

상기 (c) 단계는,

태스크 수행 주기 패턴을 기반으로 각 태스크가 수행될 태스크 수행 주기를 설정하는 과정; 및

최악 반응 시간(Worst-Case Response Time)을 분석하여 실시간성을 검증하는 과정

을 포함하는 TTA 적용 방법.

청구항 9

제 1 항 내지 제 8 항 중 어느 한 항에 있어서,

상기 (d) 단계는,

인터럽트 마스킹을 수행하는 과정;

이벤트 플래그를 구분하는 과정; 및

이벤트 폴링 메커니즘(event polling mechanism)을 적용하는 과정

을 포함하는 TTA 적용 방법.

청구항 10

프로세서를 구비한 실시간 내장형 시스템에,

(a) 대상 시스템에 대한 TTA(Time-Triggered Architecture) 적용 태스크를 추출하는 단계;

(b) TTA 적용을 위한 이벤트 및 태스크 특성을 파악하는 단계;

(c) TTA를 위한 태스크 수행 주기를 설정하고 실시간성을 검증하는 단계;

(d) 적용 대상 환경에 따라 TTA를 적용하는 단계; 및

(e) 이벤트 손실 여부 테스팅 및 시스템 테스팅을 수행하는 단계

를 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체.

명 세 서

기 술 분 야

[0001] 본 발명은 소프트웨어 구조 개선 프로세스 기술 분야에 관한 것으로, 더욱 상세하게는 안전 필수 실시간 내장형 시스템의 구조를 시간 구동 구조(TTA: Time-Triggered Architecture)로 개선하는, 실시간 내장형 시스템을 위한 TTA 적용 방법 및 상기 방법을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체에 관한 것이다.

배 경 기 술

[0002] “실시간 시스템(Real-time System)”은 실시간성을 가진 컴퓨터 시스템 혹은 컴퓨터 시스템 집합을 말하는데, 실시간성은 그 시스템이 수행해야 하는 “태스크”를 미리 알려진 “임계 시간” 내에 완료하는 특성이다. 실시간 시스템이 수행하는 태스크는 그 시스템의 목적을 달성하기 위한 것으로 모든 태스크는 특정 “이벤트”가 발생했을 때 수행될 수 있다. 이벤트는 그 이벤트를 유발하는 주체에 따라 시스템 외부에서 발생하는 “센서 이벤트”와 시스템 내부에서 발생하는 “시간 이벤트”로 구분할 수 있다. 또한, 실시간 시스템은 “스케줄러”라는 시스템 자원 분배 정책이 있어 각 작업을 임계 시간 안에 수행하는데 필요한 시스템 자원을 분배한다.

[0003] 다음으로, 이벤트 구동 구조(ETA: Event-Triggered Architecture)에 대하여 살펴보면 다음과 같다.

[0004] 순수한 ETA 기반 실시간 시스템에서 모든 시스템 행위는 감시중인 환경으로부터의 시그니피컨트 이벤트(Significant Event)에 의해 유발된다. 시그니피컨트 이벤트(Significant Event)는 그 이벤트를 처리해야 하는 소프트웨어 시스템이 감시하고 있는 시스템 외부 환경의 상태 천이를 말한다. 동일한 시스템 외부 환경이라 하더라도 소프트웨어 시스템이 감시하고 있지 않은 상태 천이는 무시된다. 대부분의 ETA 기반 실시간 시스템에서 시그니피컨트 이벤트(Significant Event)는 인터럽트 메커니즘에 의해 중앙처리장치(CPU)에 알려진다. 시그니피컨트 이벤트(Significant Event)는 두 종류로 나눌 수 있는데, 예상할 수 있는 P-이벤트(predictable event)와 예상할 수 없는 C-이벤트(chance event)가 있다.

[0005] P-이벤트는 물리적 법칙이나 현재 시스템의 상태를 미루어 볼 때 미래의 발생 시점을 알 수 있다. 예를 들어, 전선의 온도 변화는 전선에 흐른 전류의 양과 물리 법칙으로 예측할 수 있다. 그러므로 P-이벤트를 전달하거나 처리하기 위한 시스템 자원을 미리 예측하고 준비할 수 있다. 반면에, C-이벤트의 발생은 챤스 프로세스(chance process)에 의해 유발되는데, 이는 근본적으로 임의 속성을 가지므로 결정론적으로 예측할 수 없고, 다만 확률적으로만 예측할 수 있다. C-이벤트의 일반적인 예로는 컴퓨터 시스템에서 발생하는 결함(fault)이 있을 수 있다. 원자로의 파이프 파손과 같은 C-이벤트는 그와 관련된 여러 시그니피컨트 이벤트(Significant Event)를 유발하며, 이로 인해 더 많은 이벤트들이 컴퓨터 시스템에 전달될 수 있다. 컴퓨터 시스템은 명확한 처리 한계를 가지고 있으므로 일정 개수 이상의 이벤트는 처리할 수 없다. 그러므로 이와 같은 “이벤트 샤워즈(event showers)"를 버퍼링할 수 있는 중간 저장 단계와 흐름 제어(flow control)가 필요하다.

[0006] ETA 기반 시스템을 위한 운영 체제는 응용 프로그램의 요구가 있을 때마다 자원을 분배하며, 동적 자원 분배(dynamic scheduling)를 기본으로 한다. 일반적으로, 실시간 시스템의 작업 집합이 서로 의존 관계를 가지고 있

을 때, 이들이 임계 시간을 지킬 수 있는지에 대한 문제는 NP-hard(Nondeterministic Polynomial-Hard)로 알려져 있으며, 임계 시간을 지킬 수 있다고 하였을 때 타당한 자원 분배 방법을 찾는 문제 또한 NP-hard로 알려져 있다. 동적 자원 분배 문제에 대한 체계적인 해법으로 알려진 이론은 대부분 비현실적인 가정을 가지고 있어 효용성이 낮다고 할 수 있다. 일반적인 ETA 기반 시스템을 개발할 때는 작업 집합의 우선순위를 고정시켜 놓고 테스팅 과정에서 그 우선순위를 미세하게 조정하여 시스템에 걸리는 부하 패턴을 감당할 수 있게 맞춘다.

[0007] ETA 기반 실시간 시스템의 단점으로는, 특정 시스템 시나리오별로 다를 수 있는 이벤트 발생순서와 수행 시간에 따라 다른 태스크 수행순서가 동적으로 정해지며, 이 수행순서는 각 태스크의 실시간성을 보장하지 못할 수도 있다는 단점이 있다. 그리고 ETA 기반 실시간 시스템의 실시간성 보장은 다각도로 수행된 시스템 테스팅에 의해서만 가능한데, 실제 상황을 가정한 테스트 케이스는 드물게 일어나는 C-이벤트를 포함하지 못할 수 있기 때문에 충분하지 않다는 제약 사항도 가지고 있다. 드문 경우에 대한 시스템 행위 예측성 확보는 대부분 실시간 시스템 구현에 있어 가장 중요한 사항인데, ETA 기반 실시간 시스템은 시간의 순서에 따라 고정된 태스크 수행 시나리오가 없으므로 체계적인 시스템 성능 테스팅이 불가능하다. 마지막으로, ETA 기반 실시간 시스템을 시뮬레이션 혹은 테스팅할 때 가장 큰 걸림돌 중 하나는 시뮬레이션이나 테스팅에 사용한 테스트 케이스가 얼마나 실제 사용 패턴을 대표할 수 있는 것인가에 대한 보장을 할 수 없다는 것인데, 여러 실제 사례는 실제 필드에서 발생하는 시스템 작동 시나리오가 시뮬레이션이나 테스팅으로 재현하기 어렵다는 것을 시사하고 있다.

[0008] 다음으로, 시간 구동 구조(TTA: Time-Triggered Architecture)에 대하여 살펴보면 다음과 같다.

[0009] ETA 기반 분산 시스템은 어떤 이벤트 발생을 즉시 전파하여야 하지만, TTA 기반 분산 시스템에서는 현재 시스템이 관찰하고 있는 외부 환경의 상태 변화를 다른 노드들에게 빠르게 전파하는 것이 근본적인 요구사항이다. 이 상태 정보의 전파는 그 외부 환경의 특성에 따라 일정한 주기로 감시되며, 이 주기를 관찰 기준선(observation grid)이라 한다. 외부 환경 변화와 독립적으로 그 상태 변화는 일정한 주기로 컴퓨터 시스템에 갱신된다. TTA 기반 시스템에서는 특정 이벤트가 발생할 확률을 제어할 수는 없지만, 그 이벤트 발생을 인지하는 시점은 관찰 기준선에 따라 정할 수 있다. 이러한 강제된 특성 때문에 TTA 기반 실시간 시스템은 본질적으로 ETA 기반 실시간 시스템보다 예측이 가능하다.

[0010] TTA 기반 시스템을 위한 운영체제는 미리 정해진 정적 수행순서에 따라 자원을 분배하며, 시스템의 작동 상태(Operational Mode)가 여러 개 있을 경우 각 상태에 따라 다른 수행순서를 가질 수 있다. 이 수행순서를 정할 때에는 모든 작업들의 의존 관계를 고려해야 하며, 각 작업들 간의 명시적 동기화 또한 고려해야 한다. 이러한 수행순서는 컴파일 단계에서 고정되며, 실제 소프트웨어가 수행 중에는 운영체제가 단순히 테이블만을 참조하여 특정 수행 기준선(action grid)에 놓인 다음 수행 작업에 시스템 자원을 할당하게 되며, 이 수행 기준선의 간격이 해당 TTA 기반 실시간 시스템의 최소 반응시간을 결정한다. TTA 기반 실시간 시스템에서는 모든 입/출력 행위가 미리 정해진 주기에 따라 센서 혹은 엑츄에이터를 감시/작동시켜 이루어진다.

[0011] TTA 기반 실시간 시스템은 각 태스크의 수행순서와 최악 수행 시간이 미리 알려져 있으므로, 이를 이용하여 체계적인 테스팅이 가능하다는 장점이 있다. 각 태스크가 수행되는 수행 기준선과 입력 값이 제공되는 관찰 기준선은 그 밀도가 유한하므로, 이 기준선 위에서 발생할 수 있는 모든 입력 값 조합 또한 유한하여 유한한 시간 내에 테스팅을 할 수 있다. 코드 커버리지 관점에서 봤을 때 ETA 기반 실시간 시스템에서 특정 결과를 재현하기 위해 필요한 시간적/인적 자원은 동일한 TTA 기반 실시간 시스템을 대상으로 한 것보다 많다.

발명의 내용

해결하려는 과제

[0012] 이벤트 구동 구조로 설계된 실시간 시스템은 시스템의 행위를 예측하기 어려워서 시스템의 안전성을 확보하기 어려운 문제점이 있다. 예를 들어, 실시간 시스템이 사람의 생명을 다루는 의료장비 제어에 사용될 경우, 그 시스템을 구성하는 각 태스크의 실시간성이 보장됨을 테스팅으로 보이는 것은 매우 어렵다. 그러므로 적절한 프로세스를 통해 시간 구동 구조로 개선하여 각 태스크의 실시간성을 명확하게 증명할 수 있어야 한다.

[0013] 따라서 상기와 같은 종래 기술의 제반 문제점을 해결하고 상기 요구에 부응하고자 하는 것이 본 발명의 과제이다.

[0014] 따라서 본 발명은 기존 ETA 기반 실시간 소프트웨어를 TTA 기반 실시간 소프트웨어로 개선하는, 실시간 내장형

시스템을 위한 TTA 적용 방법 및 상기 방법을 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체를 제공하는 데 그 목적이 있다.

[0015] 본 발명의 목적들은 이상에서 언급한 목적으로 제한되지 않으며, 언급되지 않은 본 발명의 다른 목적 및 장점들은 하기의 설명에 의해서 이해될 수 있으며, 본 발명의 실시 예에 의해 보다 분명하게 알게 될 것이다. 또한, 본 발명의 목적 및 장점들은 특허 청구 범위에 나타낸 수단 및 그 조합에 의해 실현될 수 있음을 쉽게 알 수 있을 것이다.

과제의 해결 수단

[0016] 상기 목적을 달성하기 위한 본 발명의 방법은, 실시간 내장형 시스템을 위한 TTA(Time-Triggered Architecture) 적용 방법에 있어서, (a) 대상 시스템에 대한 TTA 적용 태스크를 추출하는 단계; (b) TTA 적용을 위한 이벤트 및 태스크 특성을 파악하는 단계; (c) TTA를 위한 태스크 수행 주기를 설정하고 실시간성을 검증하는 단계; (d) 적용 대상 환경에 따라 TTA를 적용하는 단계; 및 (e) 이벤트 손실 여부 테스팅 및 시스템 테스팅을 수행하는 단계를 포함한다.

[0017] 한편, 본 발명은, 프로세서를 구비한 실시간 내장형 시스템에, (a) 대상 시스템에 대한 TTA(Time-Triggered Architecture) 적용 태스크를 추출하는 단계; (b) TTA 적용을 위한 이벤트 및 태스크 특성을 파악하는 단계; (c) TTA를 위한 태스크 수행 주기를 설정하고 실시간성을 검증하는 단계; (d) 적용 대상 환경에 따라 TTA를 적용하는 단계; 및 (e) 이벤트 손실 여부 테스팅 및 시스템 테스팅을 수행하는 단계를 실현시키기 위한 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체를 제공한다.

발명의 효과

[0018] 상기와 같은 본 발명은, ETA 기반으로 구현된 실시간 시스템을 TTA 기반으로 개선할 수 있는 효과가 있다.

[0019] 또한, 본 발명에 따른 개선으로 인하여 실시간 시스템의 행위 예측성을 높일 수 있을 뿐만 아니라 테스팅 과정의 효율을 높일 수 있는 효과가 있다.

도면의 간단한 설명

[0020] 도 1은 본 발명에 따른 실시간 내장형 시스템을 위한 TTA 적용 방법에 대한 일실시예 흐름도,

도 2는 본 발명에 따른 대상 시스템에 대한 TTA 적용 태스크를 추출하는 과정(11)에 대한 일실시예 상세 흐름도,

도 3은 본 발명에 따른 프로파일링 과정에 대한 일실시예 상세 흐름도,

도 4는 본 발명에 따른 프로파일링 코드가 삽입된 인터럽트 핸들링 루틴의 일예시도,

도 5는 본 발명에 따른 이벤트 소스 및 발생 시간을 기록하는 프로파일링 루틴의 일예시도,

도 6은 본 발명에 따른 TTA를 위한 태스크 수행 주기를 설정하고 실시간성을 검증하는 과정(13)에 대한 일실시예 상세 흐름도,

도 7은 본 발명에 따른 적용 대상 환경에 따라 TTA를 적용하는 과정(14)에 대한 일실시예 상세 흐름도,

도 8은 본 발명에 따른 CAP#INT 인터럽트 마스킹 레지스터(1=활성화, 0=비활성화)를 나타내는 도면,

도 9는 텍사스 인스트루먼트(Texas Instrument)사의 실시간 모터 제어 DSP인 TMS320 F2810이 제공하는 타이머 인터럽트 "T3PINT"에 이벤트 폴링 메커니즘을 구현한 일예를 나타내는 도면,

도 10은 ETA 기반 실시간 시스템의 일예시도,

도 11은 TTA 기반으로 개선된 실시간 시스템의 일예시도,

도 12는 "PWM"이라는 최종 출력 변수를 가지는 ETA 기반 실시간 시스템을 TTA 기반으로 개선하고 그 출력 변수 변화를 비교한 결과를 나타내는 도면,

도 13은 H-VAD(Hybrid Ventricular Assist Device) 인공심장 시스템의 일예시도,

도 14는 심장과 상호 작용하는 H-VAD 인공심장 시스템의 일실시예 구성도,

도 15는 인공심장 시스템의 홀센서 구성을 나타내는 도면,

도 16은 기존 T3PINT 태스크와 새로운 T3PINT 태스크를 나타내는 도면,

도 17은 PWM 변화를 비교한 결과를 나타내는 도면,

도 18a 내지 도 18d는 본 발명에 따른 최악 반응 시간 계산을 수행하는 간단한 프로토타입 자바 소스코드를 나타내는 도면이다.

발명을 실시하기 위한 구체적인 내용

[0021] 상술한 목적, 특징 및 장점은 첨부된 도면을 참조하여 상세하게 후술되어 있는 상세한 설명을 통하여 보다 명확해 질 것이며, 그에 따라 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자가 본 발명의 기술적 사상을 용이하게 실시할 수 있을 것이다. 또한, 본 발명을 설명함에 있어서 본 발명과 관련된 공지 기술에 대한 구체적인 설명이 본 발명의 요지를 불필요하게 흐릴 수 있다고 판단되는 경우에 그 상세한 설명을 생략하기로 한다. 이하, 첨부된 도면을 참조하여 본 발명에 따른 바람직한 실시 예를 상세히 설명하기로 한다.

[0022] 그리고 명세서 전체에서, 어떤 부분이 다른 부분과 "연결"되어 있다고 할 때 이는 "직접적으로 연결"되어 있는 경우뿐만 아니라 그 중간에 다른 소자를 사이에 두고 "전기적으로 연결"되어 있는 경우도 포함한다. 또한, 어떤 부분이 어떤 구성요소를 "포함" 또는 "구비"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라 다른 구성요소를 더 포함하거나 구비할 수 있는 것을 의미한다.

[0023] 도 1은 본 발명에 따른 실시간 내장형 시스템을 위한 TTA 적용 방법에 대한 일실시예 흐름도로서, 단일 시스템 의료장비에 대한 TTA 적용 프로세스를 나타내고 있다.

[0024] 도 1에서는 단일 시스템으로 구성된 의료장비 제어 실시간 소프트웨어에 시간 구동 구조를 적용하는 과정을 설명한다. 도 1에 도시된 바와 같이, 총 5 과정으로 이루어져 있으며, 각 과정별로 필요한 세부 과정과 도구 등에 대해서는 후술하기로 한다.

[0025] 먼저, 대상 시스템(예: 단일 시스템 의료장비)에 대한 TTA 적용 태스크를 추출한다(11).

[0026] 이후, TTA 적용을 위한 이벤트 및 태스크 특성을 파악한다(12).

[0027] 이후, TTA를 위한 태스크 수행 주기를 설정하고 실시간성을 검증한다(13).

[0028] 이후, 적용 대상 환경에 따라 TTA를 적용한다(14).

[0029] 이후, 이벤트 손실 여부 테스팅 및 시스템 테스팅을 수행한다(15).

[0030] 다음으로, 대상 시스템에 대한 TTA 적용 태스크를 추출하는 과정(11)에 대하여 상세히 설명하면 다음과 같다.

[0031] 실시간 시스템은 태스크 단위로 나눌 수 있다. 대상 의료장비 제어 소프트웨어에서 TTA 기반으로 수행될 외부 환경을 모니터링하는 태스크와 외부 환경을 제어하는 태스크를 구분한다. 예를 들어, 모터 속도/회전 방향을 인식하는 센서나 사용자 인터페이스의 버튼에서 발생하는 인터럽트와 관련된 태스크가 여기에 해당된다. 구분된 태스크는 차후 분석을 통해 상호간 의존관계와 각 태스크가 수행되는데 필요한 시스템 자원, 수행 패턴을 도출하고 이를 기반으로 태스크 수행 스케줄을 디자인한다.

[0032] 대상 시스템에 대한 TTA 적용 태스크를 추출하는 과정(11)은, 개발자가 직접 소스코드를 보고 수행하는 코드 검사 기법과 전용 소프트웨어 분석 도구를 사용하는 데이터 플로우 분석 기법이 있다. 따라서 TTA를 적용하고자 하는 소프트웨어의 규모에 따라 효율적인 기법을 선택하는 것이 바람직하다.

[0033] 상기 코드 검사(code inspection) 기법에 대하여 살펴보면 다음과 같다.

[0034] 코드 검사는 본래 코드 개발에 관련된 개발자 4 ~ 5명이 공동으로 코드를 검토하여 버그, 일관성의 부재 등을 찾는 과정을 말한다. 대상 의료장비 제어 소프트웨어의 크기가 크지 않을 경우(예를 들어, 약 5000 라인 이하),

소프트웨어 분석 도구를 도입할 필요 없이 개발자만으로 모니터링 태스크와 제어 태스크를 구분하고 상호 의존 관계를 분석하는 것이 효율적인 경우가 많다.

[0035] 도 2는 본 발명에 따른 대상 시스템에 대한 TTA 적용 태스크를 추출하는 과정(11)에 대한 일실시예 상세 흐름도로서, 코드 검사(code inspection) 기법을 사용한 경우를 나타내고 있다.

[0036] 먼저, 이벤트를 구분한다(21).

[0037] 상기 이벤트 구분 과정(21)에서는 실시간 시스템에 공급되는 이벤트의 종류를 파악한다. 예를 들어, 센서나 버튼 이벤트가 여기에 속하며, 분산 시스템의 경우에는 네트워크 패킷도 포함한다.

[0038] 이후, 태스크 진입점을 구분한다(22).

[0039] 상기 태스크 진입점 구분 과정(22)에서는 태스크 진입점을 구분하여 각 이벤트별로 수행되는 코드를 파악한다. 대상 의료장비 제어 소프트웨어가 구동되는 하드웨어의 특징을 파악하면, 각 인터럽트 핸들링에 사용되는 전용 함수의 위치나 이름을 손쉽게 파악할 수 있다. 예를 들어, 텍사스 인스트루먼트(Texas Instrument)사에서 개발된 실시간 모터 제어 DSP(Digital Signal Processor)의 경우, 각 센서 인터럽트 루틴의 이름은 "CAPINT#"을 기본 값으로 사용하고 있으므로, 이 루틴에 사용되는 변수나 서브 루틴을 분석하면 다른 태스크와의 관계를 파악할 수 있다.

[0040] 이후, 서브 루틴을 구분한다(23).

[0041] 상기 서브 루틴 구분 과정(23)에서는 각 태스크 진입점으로부터 호출되는 서브 루틴을 파악한다.

[0042] 이후, 공유 자료구조를 구분한다(24).

[0043] 상기 공유 자료구조 구분 과정(24)에서는 태스크를 구성하는 서브 루틴들 사이에서 공유되어 사용되는 변수나 자료구조를 파악한다.

[0044] 이후, 정의-사용(Definition-Use) 관계를 파악한다(25).

[0045] 상기 정의-사용 관계 파악 과정(25)에서는 공유 자료구조와 관련된 정의-사용 관계를 파악하여 차후 태스크 수행 순서를 디자인할 때 변수 접근 순서가 올바르게 정렬될 수 있도록 한다.

[0046] 하지만, 대상 의료장비 제어 소프트웨어의 규모가 중간 이상이고 다양한 컴포넌트를 재사용하고 있는 경우, 현재 그 코드를 관리하는 개발자가 전체 시스템을 자세히 파악하기 어려우므로, 후술되는 전문 코드 분석 도구를 활용하는 것이 효율적이다.

[0047] 다음으로, 상기 데이터 플로우(data-flow) 분석 기법에 대하여 살펴보면 다음과 같다.

[0048] 데이터 플로우 분석은 소프트웨어 분석 기법 중 하나로서 소프트웨어 내에서 사용되는 변수의 정의 및 갱신 순서에 따라 소프트웨어 행위를 분석하는 것을 말하며, 도 8의 절차를 반자동 또는 자동으로 수행할 수 있다. 데이터 플로우 분석 도구로는 다양한 정적 또는 동적 분석 도구가 있으나, 도구의 적용성과 속도 면에서 유리한 정적 분석 도구(static analysis tool)가 대부분을 차지하고 있다. 데이터 플로우 분석을 통해 복잡한 대상 의료장비 제어 소프트웨어가 가진 모니터링/제어 태스크의 의존관계를 비교적 간단하게 파악할 수 있다.

[0049] 데이터 플로우 분석을 지원하는 소프트웨어 분석 도구는 이 외에도 0으로 나눔 혹은 배열 인덱스 유효성 검사 등 다양한 오류 가능성을 찾는 기능을 지원하므로, 의료장비와 같은 안전 필수 시스템 구현에 사용이 권장된다.

[0050] 위키피디아(Wikipedia)에 정리되어 있는 데이터 플로우 분석 도구 중에서 의료장비 제어 소프트웨어에 주로 사용되는 C/C++ 언어를 지원하는 도구 모음은 하기의 [표 1]과 같다.

표 1

[0051]

이름	기능
cppcheck	공개 소프트웨어, STL을 포함한 소스코드의 다양한 에러 검출에 사용 가능함
Eclipse	정적 코드 분석 기능을 포함한 통합 개발 도구

Antic	C/C++ 코드 분석 도구, 제수가 0인 나눗셈과 배열 범위 등의 오류를 검출할 수 있음
BLAST	Lazy abstraction에 기반한 C 프로그램 모델 검증 도구
Clang	정적 분석 기능을 포함한 C/C++ 컴파일러
Coccinelle	소스코드 변형과 패턴 매칭을 지원하는 정적 분석 도구
Frama-C	C 언어를 위한 정적 분석 프레임워크
Lint	최초의 C코드 정적 분석 도구
Sparse	Linux 커널 C/C++ 소스코드를 분석하기 위해 개발된 정적 분석 도구
Splint	Lint의 기능을 확장한 공개 소프트웨어
Astree	항공분야 소프트웨어에 특화된 런타임 오류 및 속성 위반 검출 정적 분석 도구
FlexeLint	Lint의 멀티 플랫폼 버전
Fluctuat	부동 소수점 연산에 특화된 런타임 오류 검출 정적 분석 도구
Green Hills Software DoubleCheck	C/C++ 언어를 위한 정적 분석 도구
Intel Parallel Studio XE	정적 보안성 검사 기능을 포함한 정적 분석 도구
LDRA Testbed	C/C++ 언어를 위한 정적 분석 도구 세트
Monodics INFER	분리 논리에 기반한 C/C++ 정적 분석 도구
Parasoft C/C++ test	정적분석, 반위 테스팅, 코드 리뷰, 런타임 오류 검출 등의 기능을 포함한 Visual Studio/Eclipse 플러그인
Polyspace	소스코드에서 특정 런타임 오류가 존재하지 않음을 검증할 수 있는 정적 분석 도구
QA-C/C++	품질 보장과 가이드라인 검토에 특화된 정적 분석 도구
Red Lizards Goanna	Visual Studio와 Eclipse 플러그인으로 개발된 정적 분석 도구
CppDepend	복잡한 코드를 분석할 수 있도록 코드 의존 관계를 가시화해주는 정적 분석 도구. Visual Studio에 내장되어 있음

[0052] 다음으로, TTA 적용을 위한 이벤트 및 태스크 특성 파악 과정(12)에 대해여 상세히 설명하면 다음과 같다.

[0053] TTA를 적용할 이벤트 및 태스크 묶음이 파악되고, 각 태스크 간 상관관계가 파악되었으면 ①대상 이벤트의 발생 주기(패턴)와 ②관련된 태스크의 최악 수행 시간, 우선순위 등의 실시간 특성, 그리고 ③태스크에 시스템 자원을 분배하는 스케줄러의 특성을 파악한다. 파악된 각 정보를 바탕으로 이벤트에 의해 수행된 각 태스크가 미리 알려진 임계 시간 이내에 종료되어 전체 시스템의 실시간성을 만족할 수 있도록 태스크 수행 주기 및 순서(스케줄러)를 디자인(기획)한다.

[0054] 태스크의 실시간 특성 중 가장 중요한 요소는 최악 수행 시간인데, 이 정보를 파악하는 기법은 크게 실제 시스템을 테스팅하면서 측정하는 프로파일링 기법과 전용 분석 도구를 사용하는 기법으로 나눌 수 있다. 전용 분석 도구를 사용하는 기법은 각 도구에 따라 상이하므로, 본 발명에서는 프로파일링 기법을 중심으로 설명하기로 한다.

[0055] 상기 프로파일링 기법에 대해여 상세히 살펴보면 다음과 같다.

[0056] 가장 기본적인 실시간 특성 파악 방법으로는, 코드에 시간 기록 코드를 삽입하여 각 이벤트의 발생 패턴 및 해당 태스크의 최악 수행 시간 등을 다양한 테스트 케이스를 통해 측정하는 방법이 있다. 이러한 방법은 최악 수행 시간 측정 도구 등과 같은 전용 분석 도구를 학습하지 않아도 되는 장점이 있으며, 실제 대상 하드웨어의 특성이나 성능이 자연스럽게 반영된 결과를 얻을 수 있는 장점이 있다.

[0057] 단점으로는 파악된 실시간 정보의 정밀도가 사용한 하드웨어 타이머에 따라 크게 제한되며, 측정된 최악 수행 시간이나 이벤트의 발생 패턴의 대표성이 수행한 테스트 케이스에 따라 좌우되므로, 개발자가 그 코드를 충분히 숙지한 경우에만 적용하는 것이 바람직하다.

[0058] 도 3은 본 발명에 따른 프로파일링 과정에 대한 일실시예 상세 흐름도이다.

[0059] 먼저, 타이머를 선택한다(31). 즉, 이벤트 발생 빈도, 주기 및 해당 태스크의 최악 수행 시간을 측정할 타이머

를 선택한다.

[0060] 일반적으로 내장형 소프트웨어를 구동하는 하드웨어(예: DSP)는 1개 이상의 하드웨어 타이머를 내장하고 있으며, 그 정밀도(clock speed, reset point)를 개발자가 제한된 범위 내에서 임의로 설정할 수 있다.

[0061] 기존에 사용되는 타이머 특성을 프로파일링 목적으로 수정하였을 때 부작용이 발생할 수 있으므로, 사전에 코드를 면밀히 분석하여 기존에 사용되지 않는 타이머를 사용하거나 혹은 기존에 사용되던 타이머를 그대로 사용하는 방법이 있다.

[0062] 이후, 코드를 삽입한다(32). 즉, 모니터링하고자 하는 각 루틴에 프로파일링 루틴을 삽입하여 태스크의 수행 주기 및 수행 시간을 저장한다.

[0063] 도 4는 본 발명에 따른 프로파일링 코드가 삽입된 인터럽트 핸들링 루틴의 일예시도이고, 도 5는 본 발명에 따른 이벤트 소스 및 발생 시간을 기록하는 프로파일링 루틴의 일예시도이다. 도 4에서 인터럽트 핸들러가 호출되었을 때 인터럽트의 종류 및 발생 시간을 기록하는 “data_Collect” 함수를 호출하는 것을 확인할 수 있다. 도 5에서 “data_Collect”는 “EvaRegs.T2CNT” 자체 타이머를 이용하여 현재 “data_Collect” 함수가 호출된 시간과 호출된 위치를 자료구조에 저장한다.

[0064] 외부 저장장치에 대한 입/출력(I/O)이 불가능한 내장형 시스템의 경우, 저장된 데이터를 일정한 주기로 외부로 전송하는 별도의 통신 루틴이 필요하다.

[0065] 이후, 정보 획득 경로를 설정한다(33).

[0066] 프로파일링을 통해 수행 중에 획득한 각 태스크의 실시간 특성 로그는 수행 중인 내장형 소프트웨어에 기록되므로, 이를 외부 인터페이스를 통해 출력하여 저장한다.

[0067] 내장형 시스템 구현을 위해 사용되는 일반적인 DSP는 복수의 통신 포트(RS-232, USB, CAN 등)를 제공하므로, 이 인터페이스를 이용하여 내장형 시스템에 누적된 로그를 외부 PC로 전달하여 저장한다.

[0068] 이후, 테스트 케이스를 작성한다(34).

[0069] 상기 테스트 케이스 작성 과정(34)은 프로파일링에서 가장 중요한 과정으로서, 각 태스크의 소스코드와 외부에서 발생하는 이벤트 특성을 분석하여 각 태스크별로 최악 수행 시간이 발생할 수 있는 테스트 케이스와, 각 이벤트별로 최대 수행 빈도 및 최소 수행 빈도가 발생할 수 있는 테스트 케이스를 작성한다.

[0070] 이때, 최악 수행 시간을 위한 테스트 케이스 작성 척도로는 테스트 케이스가 얼마나 많은 영역의 코드를 수행했는지를 의미하는 코드 커버리지를 고려할 수 있다.

[0071] 외부 이벤트의 최대 수행 빈도 및 최소 수행 빈도를 구하는 방법은 외부 이벤트의 특성에 따라 크게 달라질 수 있으므로, 상황에 따라 대표성있는 테스트 케이스를 생성한다. 예를 들어, 모터를 제어하는 내장형 시스템인 경우, 모터 센서 이벤트의 최대 수행 빈도를 구하기 위해서는 당연히 내장형 시스템을 조작하여 모터가 가장 빠르게 혹은 느리게 작동하는 경우를 만드는 테스트 케이스를 고려할 수 있다.

[0072] 이후, 상기 작성된 각 테스트 케이스를 수행한다(35).

[0073] 프로파일링을 위해 수정된 코드와 테스트 케이스 및 프로파일링 결과를 기록할 수 있는 준비가 되어 있다면, 각 테스트 케이스를 수행하여 태스크별 최악 수행 시간, 최대 수행 빈도를 특정한다.

[0074] 다음으로, 최악 수행 시간(WCET: Worst-Case Execution Time) 분석 도구 사용 기법에 대해여 상세히 살펴보면 다음과 같다.

[0075] 전술한 프로파일링 기법은 소프트웨어의 규모가 크거나 현재 개발자가 전체 소프트웨어를 이해하지 못하면 적용하기 어렵고, 적용하더라도 그 결과가 대표성이 떨어질 수 있다. 그러므로 그러한 상황에서는 여기서 소개하는 최악 수행 시간 분석 도구를 활용하는 기법을 고려하는 것이 바람직하다.

[0076] 최악 수행 시간(WCET)은 특정 소프트웨어 전체 혹은 부분을 수행하기 시작하여 종료하기까지 걸릴 수 있는 시간 중 가장 긴 시간을 의미한다. 소프트웨어의 수행 시간은 입력 값과 그 소프트웨어를 수행하는 하드웨어의 특성에 의해 결정된다. WCET를 구할 때 사용하는 입력 값의 예로는 소프트웨어의 반복문을 가장 많이 수행하거나 가장 많은 명령문을 수행하는 것을 들 수 있으며, 고려해야 할 하드웨어 특성으로는 중앙처리장치(CPU) 속도, 중

앙처리장치(CPU) 명령어 캐슁 정책, 메모리 캐슁 정책 등이 있다.

[0077] WCET 분석 기법은 크게 정적 분석 기반과 동적 분석 기반으로 나눌 수 있는데, 정적 분석 기반의 WCET 분석 기법은 대상 소프트웨어를 정적 분석하여 가장 수행 시간이 긴, 즉 가장 많은 중앙처리장치(CPU) 명령어를 수행하는 경로와 그 경로를 유발하는 입력 값을 찾는 것을 말한다. WCET 분석을 위한 다양한 방법론과 그 방법론을 구현한 도구들을 찾을 수 있으므로, WCET 분석을 하고자 하는 소프트웨어/하드웨어 특성에 따라 선택하여 적용한다. 동적 분석 기반의 WCET 분석 기법은 정적 분석에 필요한 하드웨어 프로파일이 너무 복잡하여 명세하기 어려워지면서 대두되기 시작하였으며, 최악 수행 시간을 유발하는 테스트 케이스를 작성하고 대상 소프트웨어/하드웨어상에서 수행하여 실제 최악 수행 시간을 측정한다.

[0078] 하기의 [표 2]는 WCET 분석을 수행하는 도구를 나열한 것이다.

표 2

[0079]

이름	분류	기능
aiT	정적 분석	실행 파일과 프로그램 코드 둘 다 사용하여 최악 수행 경로 및 메모리 캐슁 정책을 고려한 최악 수행 시간 분석 도구
Bound-T	정적 분석	실행 파일과 심볼 테이블을 입력으로 하여 대상 서브 루틴 혹은 함수의 최대 수행 시간을 계산함. 최대 스택 사용 정도를 분석하는 기능도 포함되어 있음. 재귀문을 포함한 프로그램은 분석할 수 없음.
Hybrid timing analysis tool	하이브리드	정적/동적 분석을 동시에 사용함. 소스코드 분석을 통해 최악 수행 경로를 찾고 그 경로로부터 테스트 케이스를 생성 및 수행한 결과로부터 실제 프로그램의 동적 수행 모델을 작성하여 최종적인 최악 수행 시간 분석에 사용함.
Chronos	정적 분석	현대식 마이크로프로세서를 정밀하게 모델링하여 일반적인 정적 분석 방식보다 정밀한 최악 수행 시간 분석을 수행함. 메모리 캐슁 정책 모델이 반영되어 있지 않은 제약사항이 있음.
Heptane	정적 분석	C 소스코드와 실행 파일을 이용한 최악 수행 시간 분석 도구. 파이프라인 모델링, 명령어 캐슁 정책 시뮬레이션에 기반한 하드웨어 프로파일링 기능을 제공함.
RapiTime	동적 분석	C 소스코드와 실행 파일을 이용한 동적 최악 수행시간 측정 도구. 최악 수행 시간 이외에 수행 시간의 분포를 측정할 수 있음.

[0080] 다음으로, TTA를 위한 태스크 수행 주기를 설정하고 실시간성을 검증하는 과정(13)에 대해여 상세히 설명하면 다음과 같다.

[0081] 태스크 수행 주기, 최악 수행 시간 등이 정해졌으면, 각 태스크가 임계 시간을 어기지 않고 수행될 수 있는지 여부를 파악하는 실시간성 검증을 수행한다. 본 발명에서는 실시간성 검증을 위한 방법 중 대표적인 최악 반응 시간 분석 방법론과 이를 수행하는 실제 프로그램을 설명하기로 한다.

[0082] 도 6은 본 발명에 따른 TTA를 위한 태스크 수행 주기를 설정하고 실시간성을 검증하는 과정(13)에 대한 일실시예 상세 흐름도이다.

[0083] 먼저, 태스크 수행 주기를 설정한다(61). 즉, 태스크 프로파일링 과정을 통해 측정한 태스크 수행 주기 패턴을 바탕으로 각 태스크가 수행될 고정된 주기를 설정한다. 이 주기를 결정하는 간단한 방법은 측정된 이벤트 발생 주기 중 가장 짧은 주기를 선택하는 방법이다. 이 주기를 이용하였을 경우에는 모든 이벤트 발생을 누락없이 파악할 수 있다.

[0084] 단, 선택된 주기가 너무 짧아 특정 태스크의 임계 시간이 미치지 못한다면, 그 이벤트를 놓치게 되는 상황이 벌어질 수 있다. 그럴 경우에는 이벤트 버퍼를 구현하거나 각 이벤트별로 다른 주기를 설정하는 기법을 취할 수 있다.

[0085] 이후, 최악 반응 시간(Worst-Case Response Time)을 분석하여 실시간성을 검증한다(62).

[0086] 최악 반응 시간 분석은 복수의 태스크가 경쟁적으로 시스템 자원을 공유하는 환경에서 특정 태스크가 수행되기 시작해서 종료되기까지 시간을 분석하여 각 태스크의 실시간성을 검증하는 방법이다. 최악 반응 시간에는 상위

우선순위 태스크 수행으로 인해 해당 태스크가 대기한 시간을 모두 포함하고 있다는 점이 최악 수행 시간과의 차이점이다.

[0087] 실시간 시스템을 이루는 모든 태스크의 최악 반응 시간이 그 태스크의 임계 시간보다 짧을 경우, 그 실시간 시스템은 실시간성을 보장한다고 할 수 있다. 틴델(Tindell)은 그의 연구 결과에서 각 태스크의 최악 수행 시간, 최대 발생 빈도, 우선순위를 미리 알고 있을 때, 최악 반응 시간을 계산하기 위한 수식을 하기의 [수학식 1]과 같이 제시하였다.

수학식 1

[0088] $$R_i = J_i + w_i + C_i$$

[0089] 상기 [수학식 1]과 같이 태스크 i의 최악 반응 시간 R_i는 지터(jitter) J_i, 대기시간 w_i, 최악 수행 시간 C_i의 합으로 정의된다.

[0090] 여기서, 대기시간 w_i는 두 부분으로 나누어 해석할 수 있는데, 첫 번째로 분석하고자 하는 태스크 i보다 우선순위가 낮은 태스크 수행으로 인해 태스크 i가 대기하는 시간 B_i와, 두 번째로 태스크 i보다 우선순위가 높은 태스크 수행으로 인해 태스크 i가 지연되는 시간으로 정의되며, 하기의 [수학식 2]와 같다.

수학식 2

[0091] $$w_i^{n+1} = B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^n + J_i + \tau_{res}}{T_i} \right\rceil C_i$$

[0092] 상기 [수학식 2]에서 hp(i)는 태스크 i보다 우선순위가 높은 태스크 집합, τ_{res}는 대상 의료장비 하드웨어의 1 cpu 사이클 시간을 의미한다. T_i는 태스크 i의 최소 주기이고, B_i는 태스크 i가 자신보다 낮은 우선순위를 가진 태스크의 수행 때문에 기다려야 하는 시간을 의미하는데, 이 시간은 스케줄러의 정책에 따라 달라질 수 있다. 대기시간 w_i는 재귀적으로 계산되며, 1번째 w_i를 0으로 두었을 때 n+1번째 w_i가 n번째 w_i와 같을 때까지 계산한다.

[0093] 최악 반응 시간 계산은 재귀적으로 이루어지므로, 수동으로 계산할 경우 시간과 노력이 많이 필요하다. 그래서 본 발명에서는 틴델(Tindell)이 사용한 WCRT 계산 알고리즘을 도 18a 내지 도 18d와 같이 구현하였다. 도 18a 내지 도 18d는 본 발명에 따른 최악 반응 시간 계산을 수행하는 간단한 프로토타입 자바 소스코드를 나타내는 도면이다. 도 18a 내지 도 18d의 소스코드는 각 태스크의 최악 수행 시간, 우선순위, 주기가 주어졌을 때 각 태스크의 최악 반응 시간을 계산한다.

[0094] 만일, 최악 반응 시간 계산 결과, 하나 이상의 태스크의 실시간성이 만족되지 않으면 각 태스크의 주기나 우선순위를 다시 검토하는 과정으로 천이한다.

[0095] 다음으로, 적용 대상 환경에 따라 TTA를 적용하는 과정(14)에 대하여 상세히 설명하면 다음과 같다.

[0096] 전술한 과정을 통해 TTA를 적용할 태스크의 실시간 특성을 파악하고 그 실시간성을 검증하였다면, 도 7의 처리 순서에 따라 TTA를 적용할 소프트웨어/하드웨어 특징에 맞춰 실제 코드를 수정 및 필요 추가 루틴을 구현하여 적용한다.

[0097] 도 7은 본 발명에 따른 적용 대상 환경에 따라 TTA를 적용하는 과정(14)에 대한 일실시예 상세 흐름도이다.

[0098] 먼저, 인터럽트 마스킹을 수행한다(71).

[0099] TTA 기반 소프트웨어는 기존 ETA 기반 소프트웨어에서 사용하던 인터럽트 메커니즘을 더 이상 사용하지 않으므로, 하드웨어 설정을 통해 이 기능을 비활성화시켜야 한다. 도 8은 본 발명에 따른 CAP#INT 인터럽트 마스킹 레지스터(1=활성화, 0=비활성화)를 나타내는 도면이다. 도 8의 "EvaRegs.EVAIMRC"는 텍사스 인스트루먼트(Texas Instrument)사의 실시간 모터 제어 DSP인 TMS320 F2810이 제공하는 인터럽트 서비스 마스킹 레지스터로서, 이 레지스터를 0으로 조작하면 실제 이벤트가 발생하더라도 인터럽트 서비스 루틴이 수행되지 않는다.

[0100] 이후, 이벤트 플래그를 구분한다(72).

[0101] TTA 기반 소프트웨어에서는 발생한 이벤트를 일정한 주기에 따라 수행해야 하므로, 각 이벤트의 발생 여부를 알아야 한다. 일반적인 실시간 제어 하드웨어의 이벤트 핸들러 하드웨어는 이를 위한 플래그 레지스터를 제공한다. 도 8의 "EvaRegs.EVAIFRC"는 텍사스 인스트루먼트(Texas Instrument)사의 실시간 모터 제어 DSP인 TMS320 F2810이 제공하는 이벤트 발생 플래그 레지스터이며, 이와 같은 이벤트 발생 플래그 레지스터를 활용하여 발생된 이벤트에 따라 주기적으로 태스크를 수행할 수 있다.

[0102] 이후, 이벤트 폴링 메커니즘(event polling mechanism)을 적용한다(73).

[0103] 발생한 이벤트에 따라 그에 맞는 태스크가 일정한 시간 주기로 수행되어야 하며, 이를 위해 이벤트 폴링 메커니즘을 추가적으로 구현하여 적용한다. 이벤트 폴링 메커니즘은 앞서 결정한 주기에 따라 각 이벤트 발생 여부를 살피고, 이벤트가 발생했을 때 그에 따른 태스크를 호출한다. 이벤트 폴링은 일정한 주기로 수행되어야 하므로, 하드웨어 타이머 이벤트를 이용하여 구현할 수 있다.

[0104] 도 9는 텍사스 인스트루먼트(Texas Instrument)사의 실시간 모터 제어 DSP인 TMS320 F2810이 제공하는 타이머 인터럽트 "T3PINT"에 이벤트 폴링 메커니즘을 구현한 일예를 나타내는 도면이다.

[0105] 각 이벤트 발생 정보 레지스터를 활용하고 있음을 도 9에서 확인할 수 있다. 만일, 이벤트 폴링이 1ms 주기로 수행되고 다른 모든 태스크가 이벤트 폴링 메커니즘에 의해 수행된다면, 전체 시스템은 1ms 주기로 이벤트 발생을 검사하고 수행하는 TTA 시스템으로 간주할 수 있다.

[0106] 상기 "71" 과정 내지 "73" 과정의 내용을 종합하였을 때 ETA 기반과 TTA 기반 이벤트 핸들링 구조를 도식적으로 비교하면 도 10 및 도 11과 같다. 도 10은 ETA 기반 실시간 시스템의 일예시도이고, 도 11은 TTA 기반으로 개선된 실시간 시스템의 일예시도로서, 일반적인 단독 실시간 시스템처럼 센서, 버튼 및 타이머 태스크로 이루어져 있으며, 각 태스크는 비 선점형 스케줄러를 통해 시스템 자원을 할당받는다.

[0107] 도 10은 인터럽트 마스킹 레지스터가 "온(ON)" 상태이기 때문에 외부에서 발생한 이벤트가 바로 이벤트 핸들러를 거쳐 해당 태스크를 유발하게 된다. 현재 내부 타이머 이벤트로 인해 T3PINT 태스크가 수행 중이므로 CAPINT4-6 태스크가 대기하고 있는 것을 확인할 수 있다.

[0108] 도 11의 TTA 기반에서는 인터럽트 마스킹 레지스터를 "오프(OFF)" 상태로 두고, T3PINT에 구현된 이벤트 폴링 메커니즘이 이벤트 발생 플래그를 주기적으로 검사하여 해당 태스크를 수행시키는 구조로 개선되어 있다.

[0109] 다음으로, 이벤트 손실 여부 테스팅 및 시스템 테스팅 과정(15)에 대하여 상세히 설명하면 다음과 같다.

[0110] TTA 기반으로 개선된 실시간 소프트웨어는 ETA에서 발견되지 않던 반응시간 증가와 모니터링 특성의 변화로 인해 기타 부작용이 발생할 수 있으므로, 테스팅을 통해 기능상의 오류가 추가적으로 발생하지 않았는지 테스팅을 수행한다.

[0111] 또한, 이벤트 발생으로 인해 갱신되는 변수를 모니터링하여 각 태스크가 발생된 이벤트를 누락없이 수행하는지 여부를 확인한다. 일예로, 기존 ETA 기반 시스템에 수행했던 테스트 케이스를 개선된 TTA 기반 시스템을 대상으로 수행하여 두 실시간 시스템의 최종 출력 변수를 비교하는 방식을 고려할 수 있다.

[0112] 도 12는 "PWM"이라는 최종 출력 변수를 가지는 ETA 기반 실시간 시스템을 TTA 기반으로 개선하고 그 출력 변수 변화를 비교한 결과를 나타내는 도면이다. 비교한 결과는 개발자와 시스템 도메인 전문가가 협의하여 적합성 여

부를 검토한다.

[0113] 다음으로, 본 발명의 기술 내용을 인공심장 시스템에 적용하는 경우를 예로 들어 그 구체적인 실시예를 살펴보면 다음과 같다.

[0114] 먼저, 본 발명의 기술 내용을 인공심장 시스템에 적용하기 전의 경우를 살펴보면 다음과 같다.

[0115] 대상 인공심장 제어 소프트웨어가 채택하고 있는 이벤트 구동 구조(ETA: Event-Triggered Architecture)는 외부의 이벤트가 발생함에 따라 대응되는 태스크를 수행하는 방식으로, 소프트웨어의 구조가 직관적이고 이벤트에 대한 반응 시간이 빠르며 확장이 용이하다.

[0116] 가) 인공심장 시스템의 하드웨어를 분석하여 살펴보면 다음과 같다.

[0117] 도 13은 H-VAD 인공심장 시스템의 일예시도이다.

[0118] "한국인공장기센터"에서 개발된 H-VAD(Hybrid Ventricular Assist Device) 인공심장 시스템은 주문 제작된 모터로 작동하는 공압식 펌프(Pneumatic Pump)를 이용하여 최대 2개의 혈액 펌프를 구동할 수 있다. 이를 사용하는 환자나 의사는 도 13에 도시된 바와 같이 버튼으로 구성된 사용자 인터페이스를 통해 피스톤의 왕복 속도나 이동 거리를 일정한 범위 내에서 조작할 수 있으며, 총 여섯 개의 버튼(작동, 정지, 증감, 제어 모드 변경)과 현재 상태를 확인할 수 있는 액정표시장치(LCD) 화면으로 이루어져 있다.

[0119] 도 14는 심장과 상호 작용하는 H-VAD 인공심장 시스템의 일실시예 구성도이고, 도 15는 인공심장 시스템의 홀센서 구성을 나타내는 도면이다.

[0120] 공압식 펌프는 펌프 피스톤에 포함된 모터가 피스톤 축을 따라 회전하며 압력판을 위/아래로 움직이며 두 개의 혈액 펌프에 필요한 공압을 발생시킨다. 혈액 펌프에 장착된 단방향 밸브가 혈류의 방향이 일정하게 유지되도록 한다.

[0121] 센서는 총 4개의 홀 센서(Hall-Effect Sensor)가 모터에 3개, 펌프 축에 1개 장착되어 있다. 모터에 설치된 홀 센서는 모터의 회전 방향 및 회전 속도를 모니터링하는데 사용되고, 펌프 축에 설치된 센서는 피스톤의 움직임을 모니터링하는데 사용된다.

[0122] DSP(Digital Signal Processor)로는 텍사스 인스트루먼트(Texas Instrument)사의 TMS320 F2810 기반의 컨트롤러를 사용한다. 4개의 센서 신호와 사용자 버튼 이벤트를 입력으로 받아 모터 출력을 계산한다.

[0123] 사용자 인터페이스는 6개의 버튼과 액정표시장치(LCD)로 이루어져 피스톤의 왕복 속도와 이동 거리를 제어할 수 있으며, 현재 H-VAD 제어 장비의 상태를 액정표시장치(LCD)의 화면을 통해 사용자에게 보여준다.

[0124] 나) 인공심장 시스템의 제어 소프트웨어를 분석하여 살펴보면 다음과 같다.

[0125] H-VAD 인공심장 시스템에 탑재된 제어 소프트웨어는 총 7개의 태스크를 통해 현재 모터와 펌프 상태를 모니터링하고, 사용자의 입력에 따라 일정한 시간 간격(주기적)으로 다음 모터 출력을 계산한다. 각 태스크의 역할은 다음과 같다.

[0126] T3PINT는 1ms 주기로 수행되는 태스크로서, 다음의 3가지 기능을 수행한다. 첫째, 현재 피스톤 위치에 따라 다음 피스톤 속도의 가속/감속/정속 여부를 결정한다. 둘째, 피스톤이 펌프 양 끝단에 도달해 정지해 있을 경우 일정시간이 지난 뒤 반대 방향으로 진행하게 한다. 셋째, 센서의 작동 상태를 주기적으로 모니터링한다.

[0127] CAPINT1는 도 15의 모터 홀 센서에 의해 발생되는 인터럽트에 의해 비주기적으로 수행된다. 피스톤이 이동 축을 지나는 시간을 이용하여 피스톤의 정상 작동 여부를 모니터링한다.

[0128] CAPINT2,3은 사용자 인터페이스의 버튼 중 펌프 속도 및 이동 거리 조절 버튼을 눌렀을 때 수행된다. H-VAD 제어 소프트웨어의 펌프 제어 모드에 따라 펌프의 속도 혹은 이동 거리를 조정한다.

[0129] CAPINT4,5,6은 도 15의 모터 내부에 장착된 모터 홀 센서에 의해 수행되는 태스크로서, 모터의 회전 방향,

속도, 피스톤의 위치 정보를 갱신한다. 피스톤의 위치가 피스톤의 이동범위 경계 값일 경우 모터를 정지시킨다.

[0130] 다음으로, 본 발명의 기술 내용을 인공심장 시스템에 적용하는 과정을 살펴보면 다음과 같다.

[0131] 가) 태스크의 실시간 특성 측정을 위한 프로파일링 소프트웨어의 설계 및 구현에 대하여 살펴보면 다음과 같다.

[0132] H-VAD 인공심장 시스템의 태스크 타이밍 측정을 위해 구현한 프로파일링 소프트웨어는 크게 태스크와 시간 정보를 수집하는 기능, 그리고 H-VAD 인공심장 시스템으로부터 수집한 정보를 시리얼 통신을 통해 호스트 PC로 전송하는 기능으로 이루어진다.

[0133] H-VAD 인공심장 시스템에 탑재된 마이크로프로세서인 TMS320 F2810 DSP에서 수집한 데이터를 저장하는 공간은 하기의 [표 3]의 L0 또는 L1 메모리로, 그 용량은 4k*16 워드이다. 수집되는 데이터의 단위 크기가 2바이트인 것과 태스크의 수행 시간 단위가 ms임을 감안한 데이터 생성 속도를 고려하였을 때, L0, L1 메모리의 용량은 자료 분석을 위해 요구되는 충분한 양의 데이터를 수집하는데 필요한 메모리 용량보다 훨씬 부족하다. 또한, 데이터의 빠른 생성 속도에 비해 H-VAD 인공심장 시스템으로부터 수집한 정보를 전송하는데 사용되는 시리얼 통신의 전송속도가 매우 느리기 때문에, 데이터가 생성될 때마다 바로 호스트 PC로 자료를 송신할 때 통신 병목 현상이 발생한다. 하기의 [표 3]은 TMS320 F2810 DSP의 메모리 구성을 나타낸다.

표 3

[0134]

메모리		용량	용도
ROM/Flash	Flash	128K * 16	프로그램 코드 저장
	OTP ROM	1K * 16	부트 코드 중요 시스템 상수 저장
SARAM (Single-Access RAM)	L0, L1	4K * 16	데이터 저장
	M0, M1	1K * 16	다른 계열 코드 호환
	H0	8K * 16	코드 저장

[0135] 이러한 제약들을 극복하기 위해 본 발명에서는 태스크의 타이밍 측정 소프트웨어가 데이터를 수집만 하는 데이터 수집 모드와 수집된 데이터를 호스트 PC로 송신만 하는 송신 모드로 나누어 동작하도록 설계한다. L0, L1 메모리의 용량을 고려하여 최대한 저장할 수 있는 데이터의 수를 지정해 놓고, 수집된 데이터의 수가 저장할 수 있는 최대 데이트의 수와 같아지면 데이터 수집을 잠시 중단하고 수집된 데이터를 호스트 PC로 송신만 한다. 그리고 그 송신이 완료되면 다시 데이터를 수집한다.

[0136] 이러한 방식은 데이터 송신 모드일 동안 발생하는 데이터를 분실한다는 단점이 있지만, H-VAD 인공심장 시스템과 같은 경우 시간에 따른 측정값 변화가 크지 않아 반복적인 측정으로 분실된 데이터를 복구할 수 있다.

[0137] 나) TTA 개선 대상 태스크 실시간 특성 분석에 대하여 살펴보면 다음과 같다.

[0138] 전술한 태스크의 시간 특성 측정을 위한 소프트웨어를 통해 H-VAD 인공심장 시스템에서 수행되는 태스크들의 시간 특성인 태스크 수행 주기와 각 태스크별 수행 시간을 측정한다.

[0139] 시스템의 예측 가능성과 안전성 확보를 위해 시스템이 최대 부하일 때를 기준으로 주기와 수행 시간을 측정하였으며, 주기는 각 태스크의 측정된 주기 중 가장 짧은 시간을 기준으로, 수행 시간은 측정된 수행 시간 중 가장 긴 시간을 기준으로 측정하였다. 또한, 본 발명에서는 시간 특성 중의 하나인 최대 수행 시간의 분석(WCETA: Worst Case Execution Time Analysis)을 위해서 기존의 정적 분석 도구를 사용하지 않고 H-VAD 소프트웨어 테스팅 관련 연구에서 사용한 최대 커버리지 달성 테스트 케이스를 사용하여 측정하였다.

[0140] 하기의 [표 4]는 측정 결과를 나타내며, 측정 결과에서는 H-VAD 인공심장 시스템에 사용된 DSP의 한계로 인해 0.01ms 이하의 시간 간격에 대해서는 측정이 불가능하였고, 이로 인해 수행 시간이 0.01ms로 나온 태스크들은 그 수행 시간이 0.01ms 미만인 것으로 표기하였다.

표 4

[0141]

태스크 명	종류	주기 (단위: ms)	수행 시간 (단위: ms)	이벤트 소스
CAPINT1	산발적 태스크	400	< 0.01	센터 홀 센서
CAPINT2	비주기적 태스크	N/A	< 0.01	버튼
CAPINT3	비주기적 태스크	N/A	< 0.01	버튼
CAPINT4	산발적 태스크	1.3	< 0.01	모터 홀 센서
CAPINT5	산발적 태스크	1.3	< 0.01	모터 홀 센서
CAPINT6	산발적 태스크	1.3	< 0.01	모터 홀 센서
T3PINT	주기적 태스크	1	0.23	타이머 인터럽트

[0142] 다) TTA 기반 소프트웨어 태스크 수행 기획 및 실시간성 분석에 대하여 살펴보면 다음과 같다.

[0143] 본 발명에서 목표로 하는 시간 구동 방식의 H-VAD 제어 소프트웨어인 TT H-VAD의 설계를 위해서는, 먼저 각 태스크를 수행하기 위한 시간 주기를 정의한다.

[0144] 전술한 각 태스크별 시간 특성 분석을 통해 상기 [표 4]에서 센서 이벤트를 포함한 모든 태스크의 발생 주기의 최소값이 1ms보다 크기 때문에 설계 기본 수행 주기를 1ms로 정의하였다. 즉, 정의된 주기 1ms마다 이벤트 발생 여부를 확인하고 이벤트가 발생한 경우에 해당 태스크를 수행하면, 모든 주기/비주기적 이벤트에 대해 대응되는 태스크를 수행할 수 있다.

[0145] 이때, 전체 시스템의 실시간성 만족 여부를 확인하기 위해 주어진 태스크들과 스케줄링 정책에 대해서 모든 경우에 대해 태스크 집합이 완료 시간 전에 종료되는지를 확인한다. 본 발명에서는 선점형 스케줄링 정책과 주기적 태스크 집합만을 가정한 WCRT 분석을 하였고, 그 결과는 하기의 [표 5]와 같다. 하기의 [표 5]에 나타난 각 태스크의 WCRT는 각 태스크의 주기, 즉 기본 임계 시간보다 짧으므로 전체 시스템의 실시간성을 만족시킴을 알 수 있다.

표 5

[0146]

태스크 명	P	B	w	C	R
CAPINT1	1	0.23	0.23	0.01	0.24
CAPINT2	2	0.23	0.24	0.01	0.25
CAPINT3	3	0.23	0.25	0.01	0.26
CAPINT4	4	0.23	0.26	0.01	0.27
CAPINT5	5	0.23	0.27	0.01	0.28
CAPINT6	6	0.23	0.28	0.01	0.29
T3PINT	7	0	0.06	0.23	0.29

[0147] 라) 이벤트 폴링 메커니즘 구현에 대하여 살펴보면 다음과 같다.

[0148] 전술한 TTA 기반 태스크 수행 기획과 측정을 통해 획득된 실시간 특성을 고려하여 TT H-VAD를 구현하기 위해서 정의된 주기인 1ms마다 이벤트 발생 여부를 확인하고 이벤트가 발생한 경우 수행되는 모니터링 코드를 구현하였다.

[0149] 이를 위해 동일한 수행 주기를 갖는 기존 H-VAD 인공심장 시스템의 태스크 중 T3PINT 태스크를 도 16에 도시된 바와 같이 수정하였다. 이벤트 발생 여부를 확인하는데 있어서, 기존 H-VAD 인공심장 시스템의 이벤트 구동 방식 구조를 위해 사용되던 인터럽트 핸들러를 비활성화하였고, 그 대신 DSP에 내장된 이벤트 발생 레지스터를 이용하여 각 이벤트 발생 여부를 확인하였다.

[0150] 기존의 H-VAD 소프트웨어 구조에서 모터가 회전함에 따라서 발생하는 이벤트(CAPINT)는 그 물리적인 특성상 회전 방향으로 순차적으로 발생하기 때문에, 제안된 TT H-VAD에서는 새로운 태스크 정의에서 조건문으로 과거 이벤트의 발생 여부를 검사한 후에 이를 모터 제어의 입력으로 사용해도 이벤트 처리의 순서가 바뀌는 현상이 발

생하지 않으며, 이에 따른 우선순위 역전 현상도 발생하지 않는다.

[0151] 다음으로, 본 발명의 기술 내용을 인공심장 시스템에 적용한 후 개선된 소프트웨어 동작 동일성 확인 및 평가에 대하여 살펴보면 다음과 같다.

[0152] 본 발명에서는 ETA 기반 소프트웨어 설계 구조를 TTA 기반으로 개선함으로써 시스템의 예측성을 향상시킬 수 있다. 단, 개선된 소프트웨어 설계 구조에 의해 제어된 모터의 동작이 기존의 소프트웨어 구조에 의해 제어된 모터의 정상적인 상황에서의 동작과 동일함을 확인하여야 한다.

[0153] H-VAD 인공심장 시스템은 모터의 센서를 통해 펌프의 상태를 확인하고 모터의 제어를 위해 DSP에 탑재된 제어 소프트웨어에서 계산된 PWM(Pulse-Width Modulation) 값을 모터의 제어 신호로 출력하는데, 본 발명에서는 동작 동일성을 확인하기 위해 각 소프트웨어 설계 구조에 의해 계산된 제어 값인 PWM의 값과 계산되는 시간을 비교하였다.

[0154] 도 17은 PWM 변화를 비교한 결과를 나타내는 도면으로, 본 발명에서 제안된 TT H-VAD가 기존 H-VAD와 비교했을 때 거의 동일한 모터 출력 특성을 보임을 확인할 수 있었다. TTA 기반으로 개선된 H-VAD 인공심장 제어 소프트웨어는 TTA 기반이 일반적으로 가지는 높은 시스템 행위 예측성과 테스팅의 용이성을 획득할 수 있다.

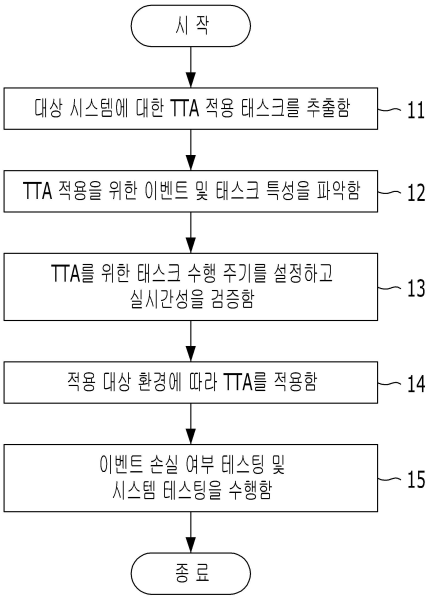
[0155] 한편, 전술한 바와 같은 본 발명에 따른 실시간 내장형 시스템을 위한 TTA 적용 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 본 발명을 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(magnetic media), CD-ROM, DVD와 같은 광기록 매체(optical media), 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical media), 및 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 상기 매체는 프로그램 명령, 데이터 구조 등을 지정하는 신호를 전송하는 반송파를 포함하는 광 또는 금속선, 도파관 등의 전송 매체일 수도 있다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용하여 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 상기 하드웨어 장치는 본 발명의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

[0156] 이상과 같이 본 발명은 비록 한정된 실시 예와 도면에 의해 설명되었으나, 본 발명은 상기의 실시 예에 한정되는 것은 아니며, 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자라면 이러한 기재로부터 본 발명의 기술적 사상을 벗어나지 않는 범위 내에서 다양한 치환, 변형 및 변경이 가능하다.

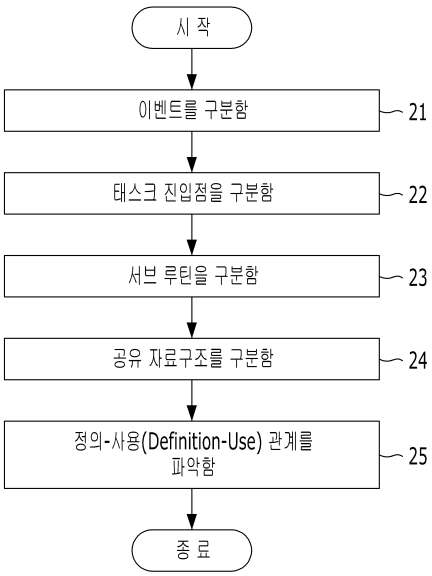
[0157] 그러므로 본 발명의 범위는 설명된 실시 예에 국한되어 정해져서는 아니 되며, 후술하는 특허청구범위뿐만 아니라 이 특허청구범위와 균등한 것들에 의해 정해져야 한다.

도면

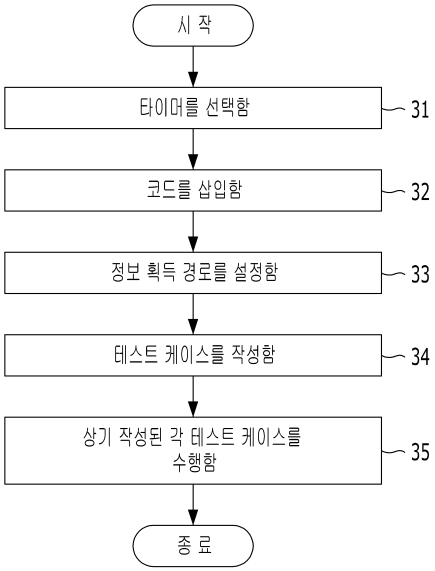
도면1


시 작
대상 시스템에 대한 TTA 적용 태스크를 추출함
11
TTA 적용을 위한 이벤트 및 태스크 특성을 파악함
12
TTA를 위한 태스크 수행 주기를 설정하고
실시간성을 검증함
13
적용 대상 환경에 따라 TTA를 적용함
14
이벤트 손실 여부 테스팅 및
시스템 테스팅을 수행함
15
종 료

도면2


시 작
이벤트를 구분함
21
태스크 진입점을 구분함
22
서브 루틴을 구분함
23
공유 자료구조를 구분함
24
정의-사용(Definition-Use) 관계를
파악함
25
종 료

도면3



도면4

```
// INT3.5
interrupt void CAPINT1_ISR(void)    // EV-A - Center Positioning with Hall Sensor
{
   data_Collect(3, 0);
   /******************/
   visited(1, 2, 1);
   /******************/
   //int16 Delta_Deviation;

   if( OneStep_Flag == TRUE ) {

       // To receive more interrupts from this PIE group, acknowledge this interrupt
       EvaRegs.EVAIFRC.bit.CAP1INT = 1;    // Clear CAP1INT flag
       PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
   }
```

도면5

```
void data_Collect(Uint16 source, Uint16 when)
{
    Uint16 tempSrc = source;
    Uint16 tempWhen = when;

    //when: 0=entry ; 1=exit

    if(systemMode == 0)
    {
        tempSrc=tempSrc<<1;

        if(tempSrc==0x4 || tempSrc==0x5)
        {
            if(CHKBIT(LCD_Flag, BIT03) == B3SET)    //if PR button is enabled
                tempSrc=tempSrc;

            if( CHKBIT(LCD_Flag, BIT02) == B2SET ) // if SL control is enabled
                tempSrc=tempSrc | 0x1;
        }

        if(counterCurrent % 4 ==0)
            counterTableSrc[counterCurrent/4]=0;

        tempSrc = tempSrc << ((counterCurrent%4)*4);

        counterTableSrc[counterCurrent/4] |= tempSrc;

        if(counterCurrent % 16 ==0)
            counterTableWhen[counterCurrent/16]=0;

        tempWhen=tempWhen<<(counterCurrent%16);

        counterTableWhen[counterCurrent/16] |=tempWhen;

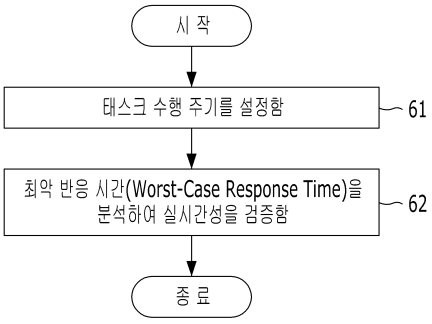
        counterTableCnt[counterCurrent]=EvaRegs.T2CNT;
```

인공심장 작동 모드 기록

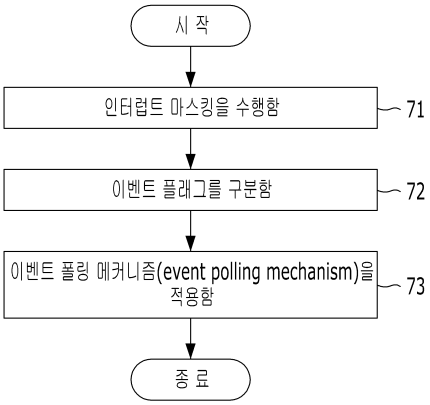
이벤트 소스 기록

이벤트 발생 시간 기록

도면6



도면7



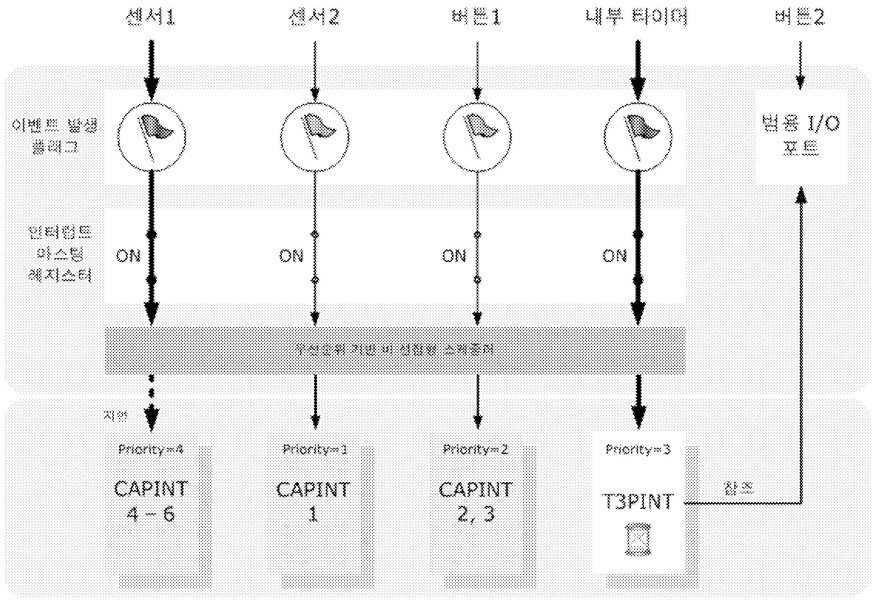
도면8

```
EvaRegs.EVAIMRC.bit.CAP1INT = 1;     // Unmask CAP1INT
EvaRegs.EVAIFRC.bit.CAP1INT = 1;     // Clear CAP1INT flag
EvaRegs.EVAIMRC.bit.CAP2INT = 1;     // Unmask CAP2INT
EvaRegs.EVAIFRC.bit.CAP2INT = 1;     // Clear CAP2INT flag
EvaRegs.EVAIMRC.bit.CAP3INT = 1;     // Unmask CAP3INT
EvaRegs.EVAIFRC.bit.CAP3INT = 1;     // Clear CAP3INT flag
```

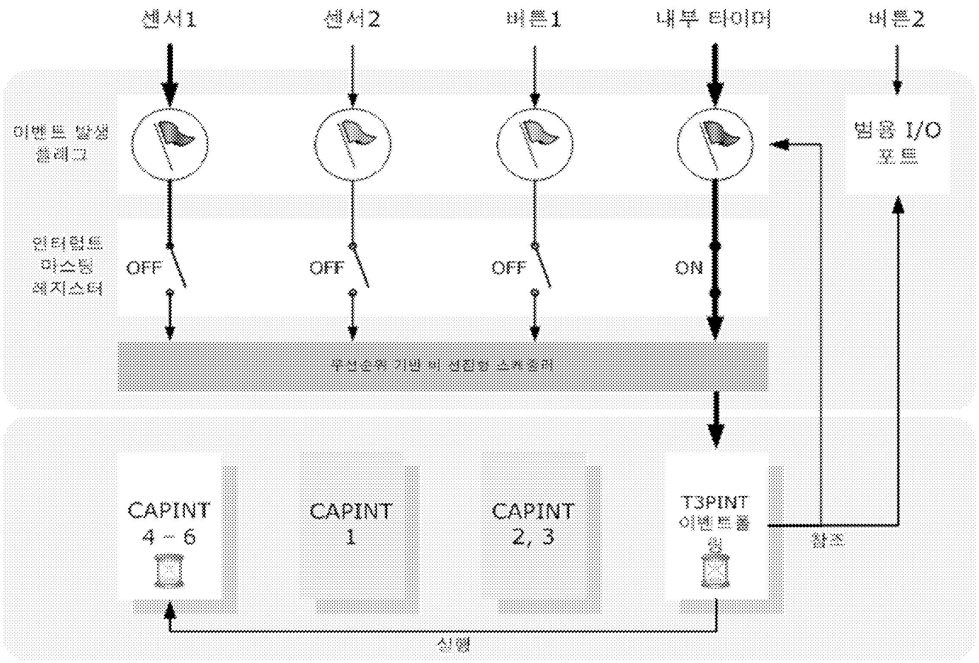
도면9

```
1:  interrupt T3PINT()
2:      if CAP1FLAG is TRUE
3:          <CAPINT1 service routines>
4:          Reset event flag CAP1FLAG to receive more interrupt
5:      endif
6:      if CAP2FLAG is TRUE
7:          <CAPINT2 service routines>
8:          Reset event flag CAP2FLAG to receive more interrupt
9:      endif
10:     ...
11:     <T3PINT service routines>
12:     Reset event flag T3FLAG to receive more interrupt
13: endinterrupt
```

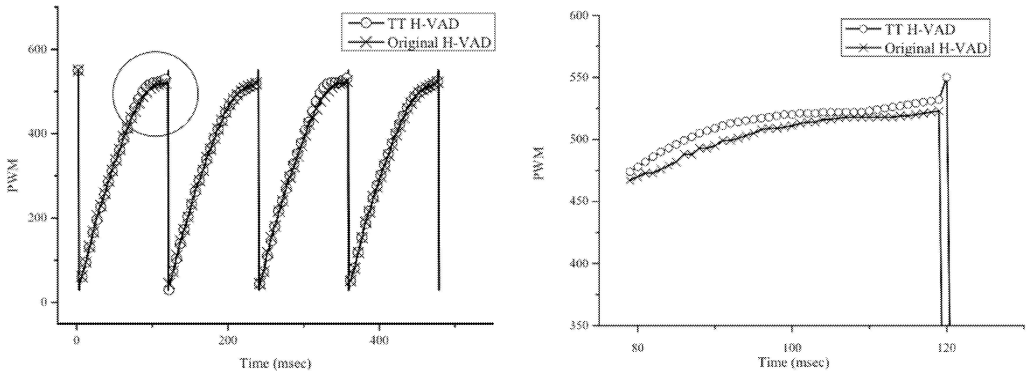
도면10


센서1
센서2
버튼1
내부 타이머
버튼2
이벤트 발생 플래그
범용 I/O 포트
인터럽트 마스팅 레지스터
ON
ON
ON
ON
우선순위 기반 비 선점형 스케줄러
지연
Priority=4
CAPINT 4 - 6
Priority=1
CAPINT 1
Priority=2
CAPINT 2, 3
Priority=3
T3PINT
참조

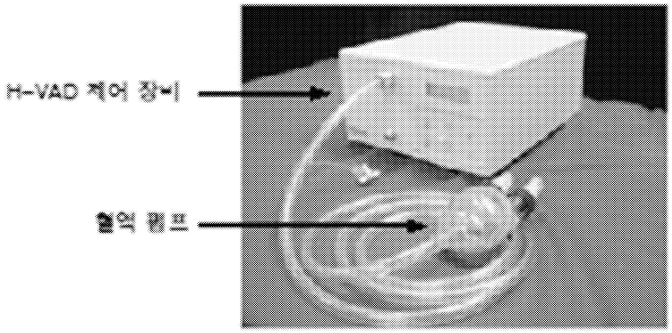
도면11


센서1
센서2
버튼1
내부 타이머
버튼2
이벤트 발생 플래그
범용 I/O 포트
인터럽트 마스팅 레지스터
OFF
OFF
OFF
ON
우선순위 기반 비 선점형 스케줄러
CAPINT 4 - 6
CAPINT 1
CAPINT 2, 3
T3PINT 이벤트를 실
참조
실행

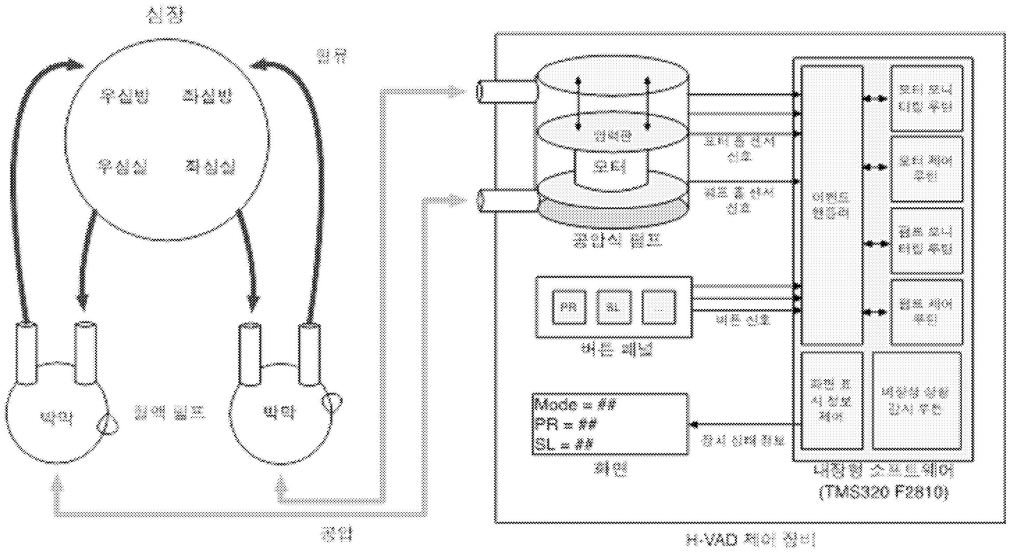
도면12



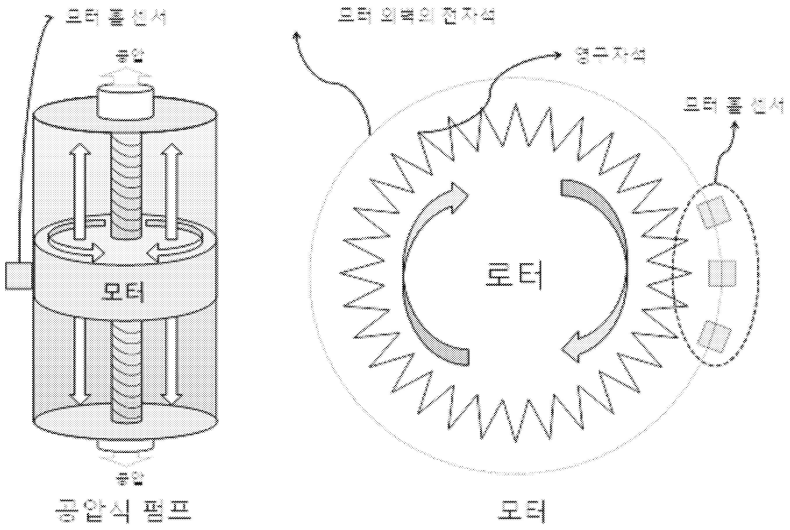
도면13



도면14



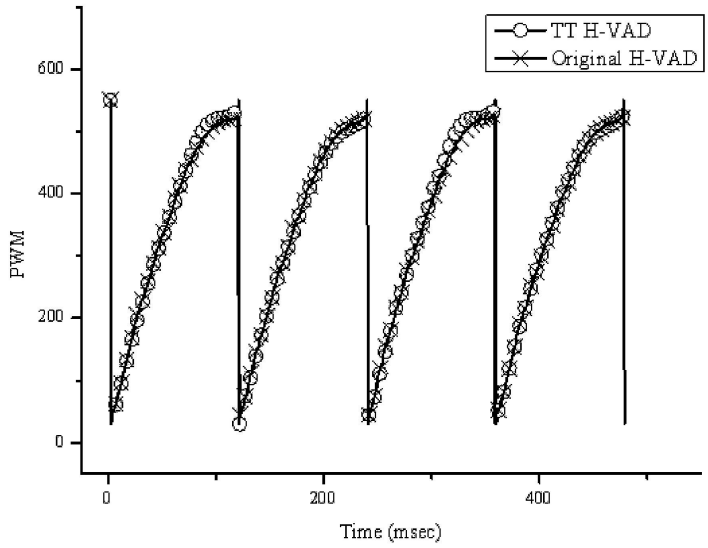
도면15



도면16

<기존 테스크 정의>	<개선된 테스크 정의>
1: interrupt CAPINT1()	1: interrupt T3PINT()
2: <service routines>	2: if CAP1FLAG is TRUE
3: Reset event flag CAP1FLAG to receive more interrupt	3: <CAPINT1 service routines>
4: endinterrupt	4: Reset event flag CAP1FLAG to receive more interrupt
5: interrupt CAPINT2()	5: endif
6: <service routines>	6: if CAP2FLAG is TRUE
7: Reset event flag CAP2FLAG to receive more interrupt	7: <CAPINT2 service routines>
8: endinterrupt	8: Reset event flag CAP2FLAG to receive more interrupt
9: …	9: endif
10: interrupt T3PINT()	10: …
11: <service routines>	11: <T3PINT service routines>
12: Reset event flag T3FLAG to receive more interrupt	12: Reset event flag T3FLAG to receive more interrupt
13: endinterrupt	13: endinterrupt

도면17



도면18a

가. 코드 1: main.java

```
import realtimeSystem.TaskSet;
public class Main {
	public static void main(String[] args){
		TaskSet m = new TaskSet();
		m.buildTaskSet("taskProp");
		m.calcWCRT(10, 0.01);
	}
}
```

나. 코드 2: Task.java

```
package realtimeSystem;
public class Task {
	private String name;
	private double pTime;
	private double eTime;
	private int p;
	private double wcrt;

	public Task(String name){
		this.name = name;
		this.pTime = 0.0;
		this.eTime = 0.0;
		this.p = -1;
		this.wcrt = -1.0;
	}
	public double getPTime() {
		return pTime;
	}
	public void setPTime(double pTime) {
		this.pTime = pTime;
	}
	public double getETime() {
		return eTime;
	}
	public void setETime(double eTime) {
		this.eTime = eTime;
	}
	public int getP() {
		return p;
	}
```

도면18b

```
        public void setP(int p) {
                this.p = p;
        }
        public double getWcrt() {
                return wcrt;
        }
        public void setWcrt(double wcrt) {
                this.wcrt = wcrt;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
}
```

다. 코드 3: TaskSet.java

```
package realtimeSystem;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
public class TaskSet {
        private Set<Task> taskSet;
                public TaskSet(){
                taskSet = new HashSet<Task>();
        }
        public int buildTaskSet(String fileName){
                //Task name, priority, execution time, period;
                File file = new File(fileName);
                System.out.println("File: " + fileName);

                BufferedReader fr;
                String readLine;
                String[] splitLine = new String[5];
                Task t;

                if(file.exists()){
```

도면18c

```
                        try{
                fr = new BufferedReader(new FileReader(file.getAbsoluteFile()));

                while((readLine = fr.readLine()) != null){
                        readLine = readLine.substring(0, readLine.length() -
1);
                        splitLine = readLine.split(",");
                        t = new Task(splitLine[0]);
                        t.setP(Integer.parseInt(splitLine[1]));


t.setETime(Double.parseDouble(splitLine[2]));


t.setPTime(Double.parseDouble(splitLine[3]));
                        taskSet.add(t);
                }
                }catch(FileNotFoundException e){
                        e.printStackTrace();
                }catch(IOException e){
                        e.printStackTrace();
                }

                System.out.printf("Total %d tasks.\n", taskSet.size());
                }
                return 0;
        }
        public Set<Task> getTaskSet() {
                return taskSet;
        }
        public void setTaskSet(Set<Task> taskSet) {
                this.taskSet = taskSet;
        }
        public int calcWCRT( int bound, double res ){
                //wi = max(c) + sigma((w+resolution)/periodi)*ci
                //Ri = wi + ci

                System.out.printf("Begin calculation with %d bound and %f
resolution.\n", bound, res);

                for(Task t : taskSet){
                //Get first term in wi.
                double maxET = 0.0;
```

도면18d

```
                for(Task tt : taskSet){
                        if((tt.getP() > t.getP()) && (tt.getETime() >= maxET))
                        maxET = tt.getETime();
                }


                //Get wi term
                double old_w = 0.0;
                double new_w = 0.0;
                double tmp = 0.0;

                for(int i = 0 ; i < bound ; i++){    //Bounded loop
                        tmp = 0.0;
                        old_w = new_w;

                        for(Task tt : taskSet){
                                if(tt.getP() < t.getP()){
        tmp = tmp + Math.ceil((old_w + res)/tt.getPTime())*tt.getETime();
                                }
                }
                new_w = maxET + tmp;
                }


                //Get Ri
                t.setWcrt(new_w + t.getETime());
                System.out.printf("R(%s) = %f(%f) + %f = %f (B=%f)\n",
t.getName(), new_w, old_w, t.getETime(), t.getWcrt(), maxET);
                }

                printResult();
                return 0;
        }
                private void printResult(){
                System.out.println("<Result>");
                for(Task t : taskSet){
                System.out.printf("%s: \t P: %d \t PT: %f \t ET: %f \t
WCRT: %f \n", t.getName(), t.getP(), t.getPTime(), t.getETime(), t.getWcrt());
                }
                System.out.println("<End>");
        }
}
```