임베디드 시스템의 실시간 Black-Box Testing 실행을 위한 신호 생성 및 입력 장치 제어 방법 구현

권진석*, 정기현*, 최경희*
*아주대학교 전자과
e-mail : ahajinco@ajou.ac.kr

Implementation of Signal Generation and Capture Module for Real-Time Black-Box Testing On Embedded System

Jin-Seok Kwon*, Ki-Hyeon Chung*, Kyoeng-Hee Choi*
*Dept. of Electronics, Ajou University

요 약

임베디드 시스템의 Black-Box 테스트 자동화를 위해 Test Script 기반으로 테스트를 진행하고 있는 툴이 많이 있다. Test Script 에 기술되어 있는 입력 값을 임베디드 시스템의 실제 신호로 생성하고 또 Test Script 에 기술된 출력 예상 값을 검증하기 위해 임베디드 시스템의 출력 신호를 읽어 들이기 위해서 일반적으로 DAQ Board 를 사용한다. 이 때 다양 한 DAQ Board 또는 다른 장치를 제어하기 Service 라는 레이어를 두어 다양한 장치에 상응하는 프로그램을 만들 수 있도록 하였다. 본 논문에서는 Test Script 실행에 있어 실시간 성을 최대한 보장하기 위해 각 Service 의 실행시간을 최소화 하는 방법에 대해 논하고자 한다. 논의를 하는데 있어 다양한 Service 중 NIDAQ Board 를 이용하여 신호를 생성하고 읽어 들이는 Service 가 모든 테스트에 있어 일반적으로 사용하게 되는 Service 이기에 이를 기준으로 Service 의 실행 시간을 최소화 하는 방법을 구현 하였다.

1. 서론

Embedded Device 의 기능은 Micro Processor 의 성능 향상에 의해 간단한 기능 요구에서 더욱더 많은 기능을 요구하게 되고 있다. 그리고 Embedded Device 의 Software 의 복잡도 역시 증가하여 테스트 항목 역시 매우 증가하였다. 이에 따라 테스트 자동화 요구가 증가였다. 이 요구를 충족 시키기 위해 테스트 자동화 도구인 Rbench 를 제작하였으며 Rbench 의 구성 중 Test Script 를 입력 받아 실제 테스트를 수행하는 도구인 Test Executor 를 에서 DAQ Device 를 제어하여 Embedded Device 의 입력을 생성하고 출력을 받아 Embedded Device 가 요구사항을 충족하는지 확인 하였다.

초기에는 테스트 장치의 요구사항에 신호입력 후 반응에 걸리는 시간이 들어가지 않아 DAQ Device 의 제어에 사용되는 시간이 중요하지 않았으나 자동차의 ECU 와 같은 장치의 테스트 요구가 생겨 DAQ Board 의 실시간 제어를 통해 ECU 를 실시간으로 제어하여 응답시간을 측정 할 수 있어야 하게 되었다. 그러나 기존에 NIDAQ board 를 제어하기 위해 만들어놓은 Service 의 경우 Service 가 수행되어 DAQ Board 를 제어하는 동안 Block 되어 DAQ Board 의 응답을 기다려야 했다. Test Script 에서 테스트를 위해 많은 신호를 생성하게 되면 첫 번째 신호 생성과 마지막 신호 생성 사이에 Test Script 에서 의도 하지 않은 Delay 시간이 생기게 된다.

이를 제거하기 위해서는 Realtime OS 를 이용하여 Test Executor 를 제작하면 실시간으로 DAQ Board 를 제어 할 수 있어 구현이 가능하다.[1] 하지만 기존에 제작되어 있는 Test Executor 는 Non Reatime OS 인 Windows 기반에서 동작하도록 만들어져 있어 Non Realtime OS 위에서 DAQ Device 의 입출력 버퍼와 PXI 인터페이스의 동기클럭, 트리거신호를 이용하여 실시간으로 신호를 생성하고 받을 수 있는 방법을 찾아보았다.[2][3] 그러나 이 방법은 테스트 시간이 길어지거나 테스트를 수행하는 데 필요한 신호의 수가 많아지면 그에 따라 입출력 버퍼의 크기가 매우 커지게 된다. 따라서 많은 입출력 신호를 사용하면서 장시간 테스트를 요하는 곳에서 필요한 입출력 버퍼의 크기가 실제 제공할 수 있는 메모리 크기보다 크게 될 경우 테스트가 불가능 해진다.

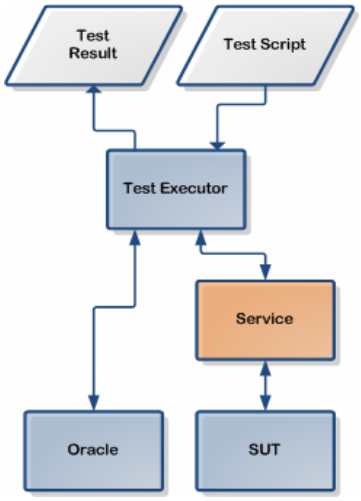
위 두 가지의 제한요소를 해결하면서 최대한의 실시간 테스트를 보장하기 위해 Service 가 실행 되는 동안 Block 되지 않는 방법과 Service 에서 DAQ Board 를 제어하기 위해 호출하는 함수의 호출 횟수를 줄이는 방법을 고안 해 보았다.

DAQ 는 National Instrument 사(이하 NI)의 NIDAQ Boad 를 사용하였으며 이를 제어하기 위해 NI 에서 제

공하는 NIDAQmx 라이브러리를 이용하였다.

2. Service 의 역할

Test Executor 에서 Test Script 를 실행하여 임베디드 시스템을 Test 하기 위해서는 Test Script 에 기술되어 있는 Logical Value 를 Physical Value 로 변환해 출력해주고 임베디드 시스템에서 출력되는 Physical Value 를 받아 모델에서 사용하는 Logical Value 로 변환해 주는 장치가 필요하다. 이 장치를 Service 라 지칭한다.

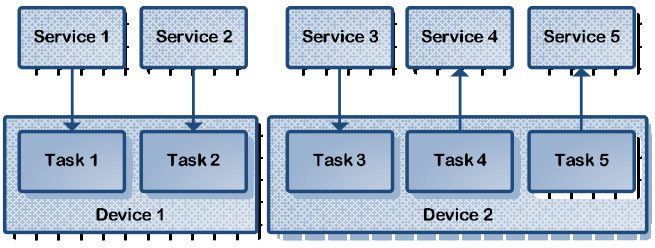


(그림 1) Test Executor 에서 Service 의 위치

Service 는 Physical Value 를 생성할 수 있는 NIDAQ Board, CAN, RS232, Resister Box 등의 다양한 Device 를 관리 한다. 본문에서는 NIDAQ Board 를 NIDAQmx 드라이버를 사용하여 Physical Value 를 생성하는 Service 를 실험하였다.

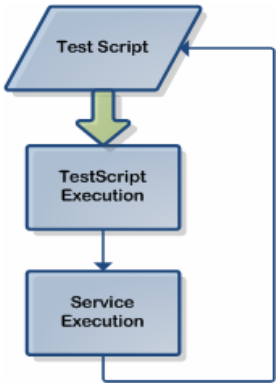
4. Blocked Service 실행과정

테스트를 수행하기 위해서는 먼저 Test Script 에 정의되어 있는 여러 가지 Value 에 matching 되어 있는 각각의 Device 를 초기화 해야 한다. NIDAQ Board 를 제어하기 위해 제공되는 NIDAQmx 라이브러리를 이용하여 각각의 Value 마다 Task 를 생성하고 초기 값을 출력한다.



(그림 2)Blocked Service 와 Task 의 관계

모든 Value 의 Device 가 초기화 되면 실제 Test 가 시작된다. Test Script 가 Test Executor 에 입력되면 Test Executor 는 Test Script 를 분석하여 해당 Service 를 실행한다. Service 실행이 완료된 후 Test Executor 는 다음 Test Script 를 분석하여 다른 Service 를 실행하게 된다.



(그림 3) Blocked Service

Service 가 실행되면서 Physical Value 를 생성하거나 입력 받기 위해 NIDAQ 장치를 Access 하는 동안 Test Executor 는 다음 Test Script 를 실행하지 못하고 Block 되어 있게 된다.

5. Blocked Service 의 문제점

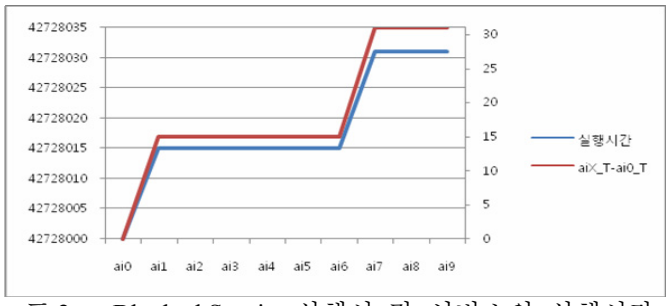
Blocked Service 를 사용하면 Test 수행과정에서 생성되는 신호의 순서를 직관적으로 알 수 있고 Physical Value 를 생성하는 NIDAQ 장치를 관리하기 쉽다는 장점이 있다. 하지만 하나의 Test Case 를 Test 하기 위해 필요한 입력이 많아 지면 모든 Physical Value 를 생성하는데 걸리는 시간이 Value 의 개수에 비례하여 시간이 누적된다. Test Case 가 장치의 실행시간을 확인하는 경우가 아닌 경우 이러한 단점은 문제가 되지 않는다. 하지만 Test Case 가 임베디드 시스템의 응답시간을 Test 하는 경우 이렇게 실행된 Test 결과는 신뢰성을 가지지 못하게 된다.

6 Blocked Service 실험 결과

이 실험에서는 한번에 생성하거나 읽어야 하는 신호가 많아지면 첫 번째 실행되는 Service 와 마지막에 실행되는 Service 의 실행 시간 차이를 보이기 위해 10 개의 Service 를 생성하고 순차적으로 실행하였다. Test Script 상에서는 각 Service 가 실행되는 필요한 시간 간격은 없으며 이것은 논리적으로는 동시에 실행되어야 함을 의미 한다. 실험에 사용한 Test Script 는 <표 1>과 같으며 이 Test Script 의 의미는 DAQ Board 의 analog input channel 0 ~ 9 까지의 신호를 읽으며 예상 결과는 0 이다. 이 실험의 결과는 <표 2>와 같다. 이 그래프에서 보는 것과 같이 첫 번째 신호를 생성하기 위해 실행된 Service 와 마지막 신호를 생성하거나 읽기 위해 실행된 Service 의 실행 시간에는 31ms 의 시간차가 생기게 된다.

```
ai0.inspect(0);
ai1.inspect(0);
ai2.inspect(0);
ai3.inspect(0);
ai4.inspect(0);
ai5.inspect(0);
ai6.inspect(0);
ai7.inspect(0);
ai8.inspect(0);
ai9.inspect(0);
```

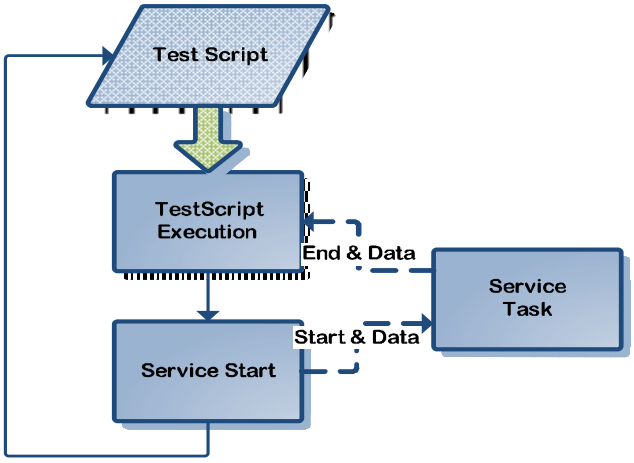
<표 1> 실험에 사용한 Test Script



<표 2> Blocked Service 실행시 각 서비스의 실행시간

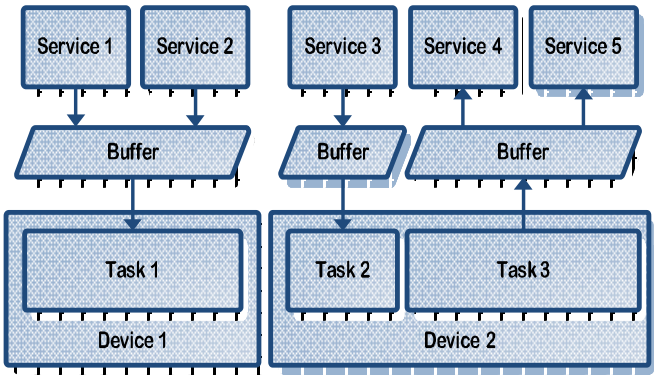
7. Non Blocked Service

Service 가 수행되는데 사용되는 시간은 Task 실행준비 시간, Task 실행 시간, Task 실행완료 처리 시간으로 구분할 수 있으며, Task 실행시간은 NIDAQ Device 의 성능에 의해 결정된다. 그리고 실험결과 실행시간 보다 준비 및 완료 처리 시간의 비중이 매우 큰 것을 알 수 있었다. 그러므로 Service 실행시간을 줄이기 위해서는 Task 실행 준비시간과 Task 실행 완료 처리 시간을 줄여야 한다. Service 가 실행되는 시간을 줄이기 위해 입력된 Task Script 를 분석하여 Concurrent 하게 실행되고 있는 Service Task 에 실행을 요청하는 구조로 변경하였다. 이 구조에서는 Test Script 를 분석하여 해당 Service Task 에 실행 요청을 Queue 에 넣고 다음 Test Script 를 실행한다. 실행 요청을 받은 Service Task 는 Queue 에 들어 있는 데이터를 출력한다. 실행이 완료되면 실행이 완료되었다는 신호를 Test Script Executer 에 알리고 실행 결과 값을 보내고 다음 요청 신호가 발생하는 시점까지 대기 한다.



(그림 4) Non Blocked Service

논리적으로 같은 시간에 실행 되어야 하는 Service 를 같은 시간에 Physical Value 로 출력하거나 입력 받는 것은 NIDAQ Device 의 ADC 또는 DAC 를 여러 channel 에서 공유하는 것과 같은 하드웨어적인 한계에 의해 불가능하지만 NIDAQ Device 의 성능을 최대한 이용하여 10ms 와 같은 단위 시간에서의 출력을 최대한 보장 하기 위해 각각의 Service 마다 따로 있는 Task 를 동일한 Device, 동일한 Function 을 가지는 Service 의 Task 를 하나로 묶어 같이 실행 하도록 하였다.



(그림 5) Non Blocked Service 와 Task 의 관계

Test Script 에 기술되어 있는 Value 를 임베디드 시스템에 입력 하는 Service1 의 실행 요청이 들어 오면 Buffer 에 있는 Service1 의 값이 갱신되고 Task1 이 실행되어 Buffer 에 있는 Service1 과 Service2 의 값이 같이 출력되게 된다. Task 가 실행되기 전에 Service2 의 실행요청이 발생하게 되면 Task 는 한번의 실행으로 두 개의 Service 를 모두 수행하게 된다.

임베디드 시스템의 상태를 확인하는 Service4 의 요청이 들어 오면 해당 요청을 Queue 에 넣고 다음 Test Script 를 수행한다. 해당 Service Task 는 Service4 와 Service5 에 해당하는 값을 모두 읽어 Buffer 에 저장하고 저장된 값 중에서 Queue 에 들어온 실행 요청을 확인 하여 해당 Service 에 값을 넘겨 준다. Service Task 가 Queue 를 확인하기 전에 Service5 의 요청이 발생하여 Queue 에 들어 왔다면 Service Task 는 한번의 실행으로 두 개의 Service 요청을 처리하게 된다.

```
ServiceExecution(Data, PendingInfo)
  EnterCriticalSection()
    Queue.Push(Data, PendingInfo)
    SendEvent()
  LeaveCriticalSection()
End ServiceExecution
```

(그림 6) Service Execution Pseudo Code

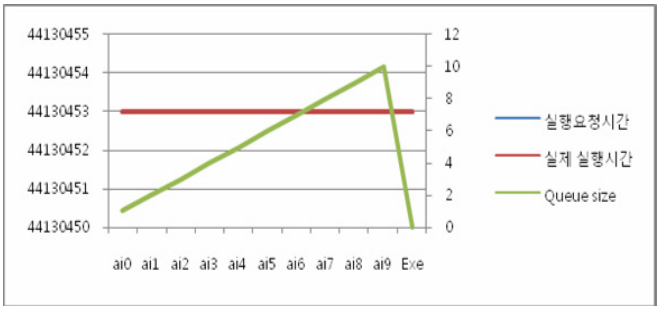
```
Thread ()
  While(1)
    WaitEvent()
    EnterCriticalSection()
      IF Queue is Empty
        LeaveCriticalSection()
        Continue
      End IF
      For i=0; i<Queue.size; i++
        Data [i] = Queue.Front()
        PendingInfo[i] = Queue.Front()
        Queue.Pop()
      End For
    LeaveCriticalSection()
    DAQmxWriteDriver (Data)
    ServiceDoneEvent (PendingInfo)
  End While
End Thread
```

(그림 7) Service Thread Pseudo Code

이 방법으로 Service 를 실행하게 되면 Test Script 에 같은 시간에 수행되어야 하는 Service 요청을 처리하는데 있어 NIDAQ Device 의 실행 요청-응답 횟수를 줄여 입출력 개수에 비례해 늘어나는 NIDAQ Device 의 실행 시간을 단축 시킬 수 있다.

8. Non-Blocked Service 실험 결과

이 실험에 서는 Blocked Service 와의 차이를 보기 위해 Blocked Service 실험에 사용한 Test Script<표 1> 를 그대로 사용하였다. 이 실험결과 Test Script 를 실행해는 Service 에서는 queue 에 실행 요청을 넣고 바로 다음 Script 를 실행하여 10 개의 요청이 queue 에 모두 들어간 후 DAQ Board 의 실행 Task 는 한번만 실행됨을 <표 3>의 그래프에서 확인 할 수 있다. Task 의 실행 횟수를 줄여 실행에 필요한 시간을 줄일 수 있게 되어 1ms 의 시간 안에 10 개의 Service 가 모두 실행 되었음을 확인 할 수 있다.

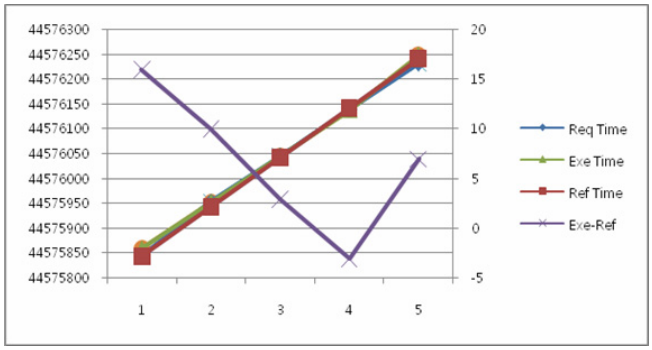


<표 3>.Non Blocked Service 실행 시 각 서비스의 실행 시간 및 Queue 크기

하지만 Test Script 에 Wait time 을 넣어 <표 4>와 같은 Test Script 를 실행 하게되면 <표 5>과 같은 결과를 볼 수 있었다.

```
ai0.inspect(0);
wait(100);
ai0.inspect(0);
wait(100);
ai0.inspect(0);
wait(100);
ai0.inspect(0);
wait(100);
ai0.inspect(0);
wait(100);
```

<표 4> Wait 를 포함한 Test Script



<표 5> <표 4>의 실행 시간 결과

Service 를 실행하고 일정 시간 후에 다시 Service 를 실행 하는 경우 중간에 Wait(100)(100ms 를기다린다.) 과 같은 script 를 추가하여 일정 시간을 기다리게 된다. 실시간 테스트를 위해서는 Wait time 이 정확하게 준수 되어야 한다. 하지만 Non Realtime OS 인 Windows 의 특성상 Service 와 Task Thread 간의 정확한 Task Switch 가 일어나게 할 수 없다. 그래서 Test Script 상의 시간과 실제 실행 시간 사이의 시간 차가 <표 5>과 같이 최대 16ms 의 시간 차가 발생 함을 볼 수 있다.

9. 결론

본 논문에서는 임베디드 시스템의 실시간 Black-Box 테스트를 위해 Rbench 의 Test Executor 를 사용하여 Test Script 를 실행할 경우 발생하는 Service 의 실행 시간 차이 문제를 해결하기 위해 Service 가 실행될 때 Block 되는 문제와 Service 실행하는데 필요한 DAQ Board 를 제어하는 함수의 호출횟수를 줄이는 방법을 구현해 보았다. 이 경우 실험결과와 같이 다수의 신호를 생성하거나 읽을 때 발생하는 처음과 마지막의 시간 차이차이가 신호의 수의 비례하여 늘어나는 문제를 해결하는 것에는 긍정적인 결과를 얻을 수 있었으나 Non Realtime OS 인 Windows 의 한계에 의해 Thread 가 실행되는 시간을 제어 할 수 있는 방법을 찾지 못해 실행 되는 시간을 제어 할 수 없었다. 차후 이 문제를 해결할 수 있는 방법을 연구하여 Test Executor 의 실시간 테스트 성능을 증가시켜야 하겠다.

참고문헌

[1] Barrera, E.; Ruiz, M.; Lopez, S.; Machon, D.; Vega, J.; “PXI-based architecture for real time data acquisition and distributed dynamical data processing” Real Time Conference, 2005. 14th IEEE-NPSS

[2] Ullrich, A., “PXI Express for Real-Time Control and High Performance Acquisition (April 2007)” Real-Time Conference, 2007 15th IEEE-NPSSChristos

[3] Stock, S.;” Implementing advanced timing and synchronization architectures in PXI systems” Autotestcon, 2005. IEEER.