한국산학기술학회논문지
Vol. 12, No. 3 pp. 1380-1386, 2011

비동기 시리얼 통신의 성능 향상을 위한 인터럽트 통합 기법

박근덕[1], 오삼권[1*], 김병국[1]
[1]호서대학교 컴퓨터공학부

An Interrupt Coalescence Method for Improving Performance of Asynchronous Serial Communication

Geun Duk Park[1], Sam Kweon Oh[1*] and Byoung Kuk Kim[1]
[1]**Division of Computer Engineering, Hoseo University**

요 약 인터럽트의 발생은 태스크의 문맥전환(context switching)을 수반한다. 이러한 문맥전환 오버헤드는, 인터럽트가 빈번하게 발생하는 경우, 임베디드 시스템의 성능을 심각하게 저하시킬 수 있다. 본 논문은 비동기 시리얼 통신에서의 빈번한 송수신 인터럽트 발생으로 인한 처리 오버헤드를 줄이기 위해, 일정한 수의 인터럽트를 누적시켜 한 번에 처리하는 인터럽트 통합(IC, interrupt coalescence) 기법을 적용한 확장 비동기 시리얼 통신 기법을 소개한다. 이 기법에 대한 성능 평가를 위해 한 바이트 단위로 송수신 인터럽트가 발생하는 기존 비동기 시리얼 통신 방식을 LN2440SBC 임베디드 보드와 uC/OS-II 상에서 구현 하여, 인터럽트 처리 소요 시간을 비교 평가한다. 평가 결과, 제안한 방식의 송수신 인터럽트 처리 소요 시간은 기존 방식에 비해, 저속(9,600 bps)의 경우, 송신은 평균 25.18% 수신은 평균 41.47%의 감소를 보이며, 고속(115,200 bps)의 경우, 송신은 평균 16.67%, 수신은 평균 25.61%의 감소를 보임으로써, 송수신 인터럽트 처리 오버헤드의 감소를 보인다.

Abstract The request of interrupt accompanies a context switching. If the interrupt is frequently requested, this overhead of context switching can reduce seriously the performance of embedded systems. In order to reduce processing overhead due to frequently requested communication interrupts at Asynchronous Serial Communication, this paper introduces the method of Expanded Asynchronous Serial Communication with the Interrupt Coalescence(IC) that accumulates a fixed number of interrupts and processes them in one time. we implement the existing Asynchronous Serial Communication that requests communication interrupts by one byte at an LN2440SBC embedded board with a uC/OS-II and compare interrupt processing time for the performance evaluation about proposed method. As a result, the communication interrupt processing time of proposed method appears in case of low speed(9,600 bps), the decline of an average 25.18% at transmission, the decline of an average 41.47% at reception. and in case of hight speed(115,200 bps), the decline of an average 16.67% at transmission, the decline of an average 25.61% at reception.

Key Words : Asynchronous Serial Communication, Interrupt Overhead, Interrupt Coalescence, Context Switching, Embedded System

1. 서론

프로세서와 연결된 장치로부터 데이터를 받아들이는 방법은 크게 폴링(polling) 또는 인터럽트 구동(interrupt-driven) 방식으로 구분할 수 있다. 빠른 응답 시간을 요구하는 시스템의 경우, 데이터의 존재 유무를 주기적으로 확인하는 폴링보다 데이터가 존재시 인터럽트를 발생시켜 해당 서비스 루틴을 수행하도록 하는 인터럽트 구동 방식이 더 적합하다[1].

인터럽트는 시스템의 장치에 의해 발생되는 신호로써,

본 논문은 2010년도 호서대학교의 재원으로 학술연구비 지원을 받아 수행된 연구임.
*교신저자 : 오삼권(ohsk@hoseo.edu)
접수일 11년 01월 12일 수정일 11년 02월 21일 게재확정일 11년 03월 10일

문맥전환을 수반한다. 이러한 문맥전환 오버헤드는 인터럽트가 빈번하게 발생하는 경우, 임베디드 시스템의 성능을 심각하게 저하시킬 수 있다.

본 논문은 비동기 시리얼 통신에서의 빈번한 송수신 인터럽트 발생으로 인한 인터럽트 처리 오버헤드를 줄이기 위해 일정한 수의 인터럽트를 누적시켜 한 번에 처리하는 인터럽트 통합(IC, interrupt coalescence) 기법[2,3]을 적용한 확장 비동기 통신 기법을 제안한다. 제안한 기법에 대한 성능평가를 위해, 한 바이트 단위로 송수신 인터럽트가 발생하는 기존 비동기 시리얼 통신 방식을 LN2440SBC 임베디드 보드[4]와 실시간 운영체제인 $uC/OS-II$[5] 상에서 구현 하여, 비교 평가 한다.

성능 평가를 위해, 통신 속도, 메시지의 크기, 그리고 송수신 인터럽트를 발생 시키는 송수신 데이터의 크기(바이트). 즉, IC 크기에 따른 송수신 인터럽트 처리 소요시간을 측정한다.

평가 결과, 제안한 방식의 송수신 인터럽트 처리 소요시간은 기존 방식에 비해, 저속(9,600 bps)의 경우, 송신은 평균 25.18% 수신은 평균 41.47%의 감소를 보이며, 고속(115,200 bps)의 경우, 송신은 평균 16.67%, 수신은 평균 25.61%의 감소를 보인다.

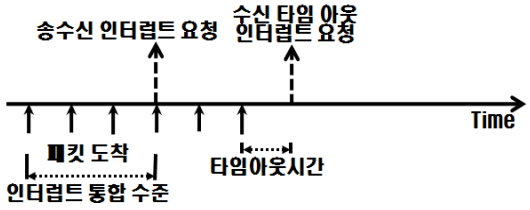
본 논문의 구성은 다음과 같다. 2장은 인터럽트 통합 기법을 소개한다. 3장은 기존 비동기 통신 모듈을 설명하고, 4장은 인터럽트 통합 기법을 적용한 확장 비동기 통신 모듈을 제안한다. 5장에서는 이 두 모듈의 성능 평가 및 분석을 하며, 마지막으로 6장에서 결론을 맺는다.

2. 인터럽트 통합 기법

인터럽트는 현재 실행중인 태스크를 선점하고 인터럽트 처리 작업을 수행하며, 이후 다시 태스크로의 문맥전환을 일으킨다. 따라서 잦은 인터럽트 발생은 실제 태스크들이 동작할 시간을 줄이게 되며, 인터럽트 발생 빈도가 프로세서의 처리 속도보다 높을 경우, 태스크들이 프로세서의 사용시간을 할당 받지 못해 실행되지 않는 라이브록(livelock) 현상[6]을 발생시키기도 한다.

IC 기법은 라이브룩 현상을 막기 위해 고속 네트워크 환경에서 사용되는 방법으로, 패킷이 도착할 때마다 인터럽트를 발생시키지 않고 일정한 양의 패킷이 도착하거나 시간이 경과 했을 때 인터럽트를 발생시킴으로써, 패킷 처리를 위한 인터럽트 발생 횟수를 줄여 인터럽트 처리 오버헤드를 감소시키는 기법이다. 이 기법을 사용하기 위해서는 통신 하드웨어 장치가 추가적인 버퍼를 유지하고, 정해진 시간이나 패킷 수에 따라, 인터럽트를 발생시키는 것을 관리해야 한다.

그림 1은 일정한 양의 패킷이 도착할 경우 인터럽트를 발생 시키는 IC 기법을 보여준다. IC의 크기는 인터럽트를 발생 시키는 패킷의 수를 의미하며, 정해진 패킷 수, 즉 IC의 크기와 같은 패킷의 수가 버퍼에 도착했을 경우, 송수신 인터럽트를 발생 시킨다. 만일 수신시 IC의 크기보다 작은 패킷의 수를 수신 받고, 일정한 시간(타임아웃 시간)까지 새로운 패킷 수신이 없다면 전송이 완료 된 것으로 보고, 수신 타임아웃 인터럽트를 발생시킨다.

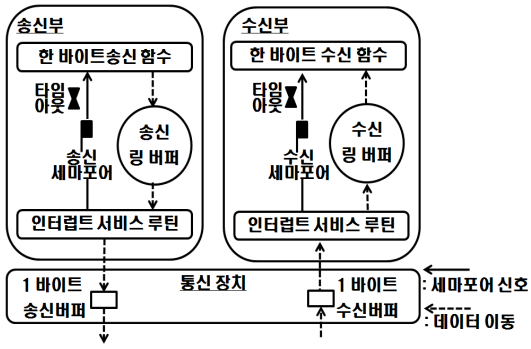


[그림 1] 인터럽트 통합(IC) 기법

IC 기법을 비동기 시리얼 통신에 적용할 경우, 전송 데이터는 패킷 단위가 아닌 시작 비트, 데이터 비트, 정비 비트, 패리티 비트로 이루어진 프레임 단위로 이루어지며, 이 경우, IC의 크기는 인터럽트를 발생 시키는 프레임의 수를 의미한다.

3. 한 바이트 단위 송수신 인터럽트 기반의 기존 비동기 시리얼 통신 모듈

그림 2는 Jean J. Labrosse가 저술한 Embedded Systems Building Blocks의 비동기 시리얼 통신 모듈을 보여준다[7]. 인터럽트 요청 조건은 표 1과 같다.



[그림 2] 기존 비동기 시리얼 통신 모듈

[표 1] 기존 방식의 인터럽트 요청 조건

송신 인터럽트	- 송신 버퍼가 비었을 경우
수신 인터럽트	- 수신 버퍼에 한 바이트 데이터가 수신되었을 경우

기존 모듈은 각각 1개의 송수신 링 버퍼를 가진다. 선입선출 방식의 환형 큐 구조인 링 버퍼는 태스크와 ISR 간의 공유메모리(shared memory)로 카운팅 세마포어(counting semaphore)를 이용해 송신 링 버퍼의 빈 공간과 수신 링 버퍼의 수신 데이터의 유무를 확인한다.

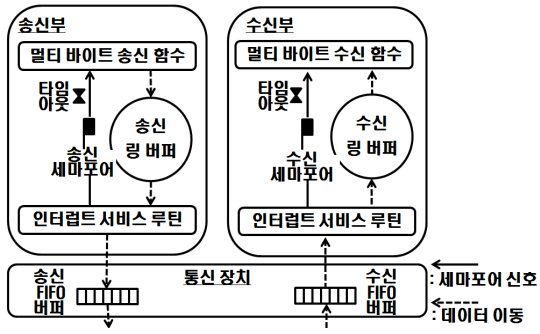
태스크 레벨에서 호출 되는 송수신 함수는 한 바이트 단위로 송수신을 수행한다.

송신 함수는 링 버퍼의 빈 공간이 존재하기를 대기(세마포어 대기)하며, 일정한 시간(세마포어 타임아웃)이 지나도록 빈 공간이 존재 하지 않는다면 종료한다. 빈 공간이 존재한다면, 한 바이트의 데이터를 링 버퍼에 복사 하고, 송신 인터럽트를 활성화함으로써, 송신 인터럽트 발생을 유도한다. ISR에서는 한 바이트의 데이터를 링 버퍼에서 송신 버퍼로 복사하고, 송신 함수에게 세마포어 신호를 보내, 링 버퍼에 한 바이트의 빈 공간이 추가되었음을 알린다.

수신부는 한 바이트의 데이터가 수신 될 경우, ISR에서 수신 버퍼에 수신된 한 바이트의 데이터를 링 버퍼로 복사하고, 수신 함수에게 세마포어 신호를 보내 링 버퍼에 한 바이트의 데이터가 저장되었음을 알린다. 수신 함수에서는 일정한 시간(세마포어 타임아웃) 동안 링 버퍼에 데이터가 삽입되기를 대기(세마포어 대기)를 하며, 링 버퍼에 데이터가 존재시, 한 바이트를 읽어 온다. 따라서 송수신시 매 바이트 마다 인터럽트가 요청 되고, 인터럽트 처리 및 문맥전환을 수반한다.

4. 인터럽트 통합 기법을 적용한 비동기 시리얼 통신 모듈

IC 기법을 적용한 송수신 모듈은 기존 비동기 통신을 기반으로 설계 했으며, 그림 3과 같이 소프트웨어 링 버퍼와 하드웨어 FIFO 버퍼를 사용하는 인터럽트 기반의 이중 버퍼링 구조를 가진다. 송수신 함수는 멀티 바이트 단위로 송수신이 이루어지며, 링 버퍼의 빈 공간 및 데이터의 존재 유무는 기존 비동기 시리얼 통신 모듈과 같이 카운팅 세마포어를 이용하여 확인한다.



[그림 3] IC 기법을 적용한 비동기 시리얼 통신 모듈

다음은 송신 함수의 의사코드를 보여준다. 인자로 송신할 메시지와 메시지의 크기, 타임아웃 시간을 입력 받는다.

```
Send(Tx_메시지, 메시지_크기, 타임아웃시간) {
  for(1 to 메시지_크기) {
    Tx_링버퍼_빈공간_존재_대기(타임아웃시간);
    if( 타임아웃 )  return;
    모든_인터럽트_비활성화;
    Tx_링버퍼 = 전송할_다음_바이트(Tx_메시지);
    if(Tx_링버퍼_데이터_크기>=Tx_IC_크기)
      송신_인터럽트_활성화;
    else if(링버퍼_저장한_전체_데이터_크기==메시지_크기)
      Tx_인터럽트_활성화;
    모든_인터럽트_활성화;
  }
}
```

송신 함수는 송신 링 버퍼에 하나 이상의 빈공간이 생기기를 대기하며, 타임아웃 시간 동안 빈공간이 존재하지 않을 경우 복귀한다. 빈 공간이 존재 할 경우, 전송하려는 한 바이트의 데이터를 링 버퍼에 복사하고, 링 버퍼의 데이터 크기가 송신 IC의 크기 이상이거나 전송할 메시지를 모두 링 버퍼에 복사한 경우에 송신 인터럽트를 활성화 한다. 송신 링 버퍼는 송신 태스크와 ISR과의 공유 메모리로 링 버퍼의 접근에 대한 상호 배제를 위해, 모든 인터럽트를 비활성화, 활성화 하며, 이와 같은 작업을 송신할 데이터의 크기만큼 실행한다.

다음은 송신 ISR의 의사코드를 보여준다. 송신 링 버퍼에 데이터가 존재할 경우, 한 바이트의 데이터를 읽어와 송신 FIFO 버퍼에 복사하고, 태스크에게 송신 링 버퍼에 빈 공간이 추가되었음을 알리기 위해, 세마포어 신호를 보낸다. 이와 같은 작업을 FIFO 버퍼의 오버플로우

(over flow)를 예방하기 위해 송신 링 버퍼의 데이터 크기와 송신 FIFO 버퍼의 빈 공간 크기 중 더 작은 크기만큼 실행한다. 송신 링 버퍼에 데이터가 존재하지 않는다면, 송신 인터럽트를 비활성화 한다.

```
TxISR() {
  if(Tx_링버퍼_데이터_존재) {
    실행횟수=MIN(Tx_링버퍼_데이터_크기,
                 Tx_FIFO버퍼_빈공간_크기);
    for(1 to 실행횟수) {
      Tx_FIFO버퍼=전송할_다음_바이트(Tx_링버퍼);
      Tx_세마포어로_신호_전송;
    }
  }
  else Tx 인터럽트 비활성화;
}
```

다음은 수신 ISR의 의사코드를 보여준다. 수신 FIFO 버퍼에서 한 바이트를 읽어와 수신 링 버퍼에 저장하고, 태스크에게 데이터의 수신을 알리기 위해, 세마포어 신호를 보낸다. 이와 같은 작업을 링 버퍼의 오버플로우를 예방하기 위해 링 버퍼의 빈 공간크기와 수신 FIFO 버퍼의 데이터 크기 중 더 작은 크기만큼 실행한다.

```
RxISR() {
  실행횟수=MIN(Rx_링버퍼_빈공간_크기,
               Rx_FIFO버퍼_데이터_크기);
  for(1 to 실행횟수) {
    Rx_링버퍼=수신한_다음_바이트(Rx_FIFO버퍼);
    Rx_세마포어로_신호_전송;
  }
}
```

다음은 태스크 레벨에서 호출하는 수신 함수의 의사코드이다. 인자로 수신된 데이터를 저장할 수신 버퍼, 수신 받을 데이터의 크기, 타임아웃 시간을 입력 받는다.

수신 함수는 수신 링 버퍼에 데이터가 한 바이트 이상 삽입되기를 기다리며, 타임아웃 시간 동안 데이터가 삽입되지 않는다면 복귀한다. 수신 링 버퍼에 데이터가 존재한다면 한 바이트의 데이터를 읽어와 수신버퍼에 저장한다. 링 버퍼의 접근에 대한 상호 배제를 위해, 모든 인터럽트를 비활성화, 활성화 하며, 이와 같은 작업을 수신 받을 데이터의 크기만큼 반복한다.

```
Recv(수신버퍼, 수신메시지크기, 타임아웃시간) {
  for( 1 to 수신_메시지_크기 ) {
    Rx_링버퍼_데이터_삽입_대기(타임아웃시간);
    if( 타임아웃 ) return;
    모든_인터럽트_비활성화;
    수신버퍼=수신_바이트(Rx_링버퍼);
    모든_인터럽트_활성화;
  }
}
```

5. 성능평가

본 논문은 구현을 위해 ARM9 기반의 S3C2440A 마이크로프로세서가 내장된 LN2440SBC 임베디드 보드와 우선순위 기반의 선점형 실시간 운영체제인 uC/OS-II를 이용하며, 비동기 시리얼 통신으로는 RS-232C 시리얼 통신을 사용한다. RS-232C 시리얼 통신의 한 프레임의 구성은 시작비트 1 bit, 데이터 비트 8 bit, 정지비트 1 bit, 총 10 bit 로 설정한다. 따라서, 초당 최대 전송 메시지 크기는 “보오레이트 / 10” 바이트로 계산할 수 있다.

S3C2440A의 UART는 64바이트의 송수신 FIFO(first in first out) 버퍼와 송신 0, 16, 32, 48 바이트, 수신 1, 8, 16, 32 바이트의 IC 크기를 설정 할 수 있다. 송수신 인터럽트 요청 조건은 표 2와 같다.

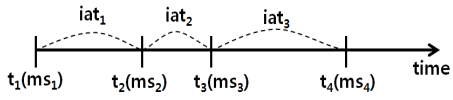
[표 2] 송수신 인터럽트 요청 조건

송신 인터럽트	송신 FIFO 버퍼가 비었거나, 데이터 크기가 IC 크기 이하인 경우
수신 인터럽트	수신 FIFO 버퍼의 데이터 크기가 IC 크기와 같거나, 3 워드(word)의 데이터를 수신 받을 시간 동안 수신 데이터가 없을 경우(타임아웃)

성능 평가는 통신 속도, 메시지 크기, 송수신 IC 크기에 따른 송수신 인터럽트 발생 횟수와 송수신 인터럽트 처리 소요시간을 측정한다.

통신 속도는 고속의 115,200 bps와 저속의 9,600 bps를 검사하고, 전체 메시지의 크기는 통신 회선의 최대 사용량의 25 %, 50 %, 75 %, 100 %로 생성하며, 1회 생성 메시지 크기는 115,200 bps에서는 1 ~ 1,000 바이트로, 9,600 bps는 1 ~ 100 바이트로 생성한다. 송신 부는 한 개의 송신 태스크를, 수신 부는 한 개의 수신 태스크를 실행하며, 두 태스크의 주기는 10 ms로 설정하였다.

그림 4와 같이 성능평가를 위한 송신 메시지의 생성 시간 간격(iat_n)과 생성된 메시지의 크기(ms_n)는 램덤(random)으로 생성되며, 지수 분포(exponential distribution)를 따른다[8].



[그림 4] 메시지 생성 시간 간격과 크기 관계

성능평가를 위해 다음과 같은 요소들을 측정하고, 식 (1)과 같이 전체 인터럽트 처리 소요시간을 계산한다.

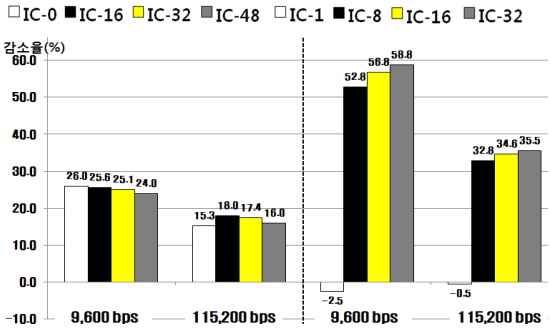
※ 전체 송수신 인터럽트 처리 소요시간 = Total_ISR_Exec_Time + Int_CtxSw_Time x Tx/Rx_Int_Ctr + CtxSw_Time x CtxSw_Ctr 식 (1)

- Tx/Rx_Int_Ctr : 송수신 인터럽트 횟수
- CtxSw_Ctr : 태스크간 문맥전환 횟수
- Total_ISR_Exec_Time : 전체 송수신 ISR 실행 시간
- Int_CtxSw_Time : 인터럽트에 의한 문맥전환 시간
- CtxSw_Time : 태스크간 문맥전환 시간

시간 측정은 PWM(pulse width modulation) 타이머를 이용하며, ARM 어셈블리어로 작성된 문맥전환 관련 부분은 실시간 운영체제의 특성상 실행 시간이 항상 일정하기 때문에, 실행 사이클의 횟수로 실행 시간을 계산한다[9]. 성능평가를 위한 송수신은 5분 동안 실행하며, 초당 평균 인터럽트 처리 오버헤드를 구한다.

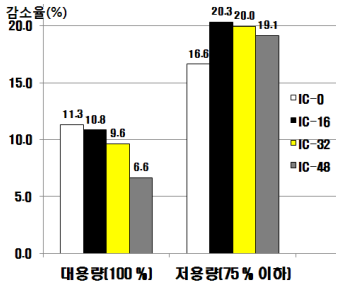
그림 5는 기본 모듈에 대한 확장 모듈의 송수신 IC 크기에 따른 평균 인터럽트 처리 소요 시간의 감소율을 보여준다. 저속의 9,600 bps에서는, 송신은 평균 25.18% 수신은 평균 41.47%의 감소를 보이며, 고속의 115,200 bps에서는, 송신은 평균 16.67%, 수신은 평균 25.61%의 감소를 보임으로써, 송수신 인터럽트 처리 오버헤드의 감소를 보인다.

또한, a에서 보이는 바와 같이, 송신 인터럽트 처리 소요 시간의 감소율은 송신 IC 크기와 반비례함을 볼 수 있다. 이는 링 버퍼에 데이터를 저장하는 빠른 CPU 속도와 데이터를 읽어가는 느린 통신 속도의 차이로 인해 링 버퍼와 FIFO 버퍼에 데이터가 쌓이게 되고, FIFO 버퍼가 다 찬 시점부터는 FIFO 버퍼의 데이터의 크기와 IC의 크기가 같을 경우 송신 인터럽트가 발생함으로, 한 송신 인터럽트 발생 당 "FIFO 버퍼의 크기(64 바이트) - IC의 크기"의 데이터가 전송되기 때문이다.



a. 송신 b. 수신

[그림 5] 송수신 IC 크기에 따른 평균 인터럽트 처리 소요 시간 감소율 비교



[그림 6] 메시지 크기에 따른 송신 IC 크기별 평균 인터럽트 처리 소요 시간 감소율 비교

그러나, 115,200 bps에서는 송신 IC 크기가 0 바이트인 경우, 더 큰 크기의 IC인 경우보다 평균 감소율이 더 낮음을 보인다. 그림 6에서 보이는 바와 같이, 전송 메시지 크기가 대용량일 때, 송신 IC 크기가 0 바이트 인 경우, 가장 높은 감소율을 보이나, 저용량의 경우, 가장 낮은 감소율을 보이기 때문이다.

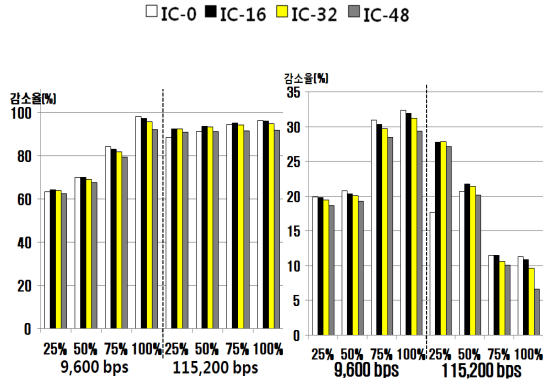
저속의 9,600 bps에서는 링 버퍼에서 데이터를 읽어가는 통신 속도가 느리므로, 영향을 많이 받지 않으나, 고속의 115,200 bps에서는 평균 메시지의 크기가 작아 질 수록 링 버퍼에 송신 메시지가 쌓이는 현상의 발생이 줄어들게 되고, 한 송신 인터럽트 당 전송 할 수 있는 데이터의 크기가 감소하며, 인터럽트의 발생 횟수와 처리 소요 시간이 늘어나기 때문이다.

수신 인터럽트는 수신 FIFO 버퍼의 데이터 크기가 수신 IC 크기와 같을 경우에 발생함으로, 수신 인터럽트 처리 소요 시간의 감소율은 그림 7의 b와 같이, 수신 IC 크기가 증가할 수 록 증가하는 것을 볼 수 있다. 수신 IC 크기가 1 바이트 경우 수신 인터럽트 발생 횟수의 감소율

이 0 %를 보인다. 이는 FIFO 버퍼에서 데이터를 읽어가는 CPU의 속도가 데이터를 저장하는 통신 속도에 비해 현저히 빠르므로, 64 바이트 중 한 바이트만을 사용하게 된다. 따라서 기존 비동기 시리얼 통신 모듈과 같은 횟수의 인터럽트가 발생하고, 기존 모듈에 비해 확장 모듈의 ISR에서는 더 많은 작업을 실행하기 때문에, 음수의 감소율을 보인다.

1 바이트의 수신 IC를 제외한 평균 감소율의 최소와 최대의 차이는 약 4 % 이내로 큰 차이는 없다. 즉, 인터럽트 통합 기법을 적용할 경우, 다소 높은 송수신 인터럽트 처리 오버헤드의 감소를 보이나, 통합 크기에 따라서는 그 차이가 작음을 볼 수 있다.

그림 7과 그림 8은 메시지 크기에 따른 기존 모듈에 대한 확장 모듈의 송수신 인터럽트 발생 횟수와 처리 소요시간의 감소율을 보여준다. X 축의 25 % ~ 100 %는 통신 회선의 최대 전송량의 백분율로, 메시지의 크기를 의미하고, Y 축은 감소율을 의미한다.



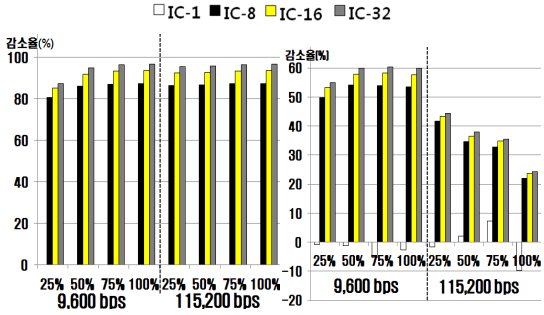
a. 횟수 감소율 b. 처리소요시간 감소

[그림 7] 메시지 크기에 따른 송신 인터럽트 횟수 및 처리 소요시간 감소율 비교

그림 7에서 보이는 바와 같이, 송신 인터럽트 횟수의 감소율은 초당 평균 메시지 크기가 증가할 수 록 증가를 보인다.

하지만, 인터럽트 처리 소요시간의 경우, 저속의 9,600 bps에서는 초당 평균 메시지 크기가 증가 할 수 록 감소율이 증가하고, 고속의 115,200 bps에서는 감소율이 감소하는 것을 볼 수 있다. 이는 대용량의 메시지 전송시, 인터럽트 처리 소요 시간의 대부분을 소요하는 메모리 접근 시간이, 캐쉬(cache)로 인해 급격한 감소를 보임에 따라, 기존 모듈의 인터럽트 처리 소요 시간의 증가 폭이 낮아지기 때문이다. ARM9 프로세서는 16K의 데이터/명령어 캐쉬와 쓰기 버퍼(write buffer)를 지원한다[9].

수신 인터럽트 횟수의 감소율은 그림 8에서 보이는 바와 같이, 메시지 크기가 증가할 수 록 소폭의 증가를 보인다. 그러나 수신 인터럽트 처리 소요 시간의 감소율은 송신의 경우와 같은 이유로 메시지 크기가 증가 할 수 록 감소함을 볼 수 있다.



a. 횟수 감소율 b. 처리소요시간 감소율

[그림 8] 메시지 크기에 따른 수신 인터럽트 횟수 및 처리 소요시간 감소율 비교

6. 결론 및 향후 계획

본 논문은 비동기 시리얼 통신에서 빈번한 송수신 인터럽트 발생으로 인한 처리 오버헤드를 줄이기 위해, 일정한 수의 인터럽트를 누적시켜 한 번에 처리하는 인터럽트 통합(interrupt coalescence) 기법을 적용한 확장 비동기 통신 모듈을 제안하고, 성능평가를 통해 인터럽트 처리 오버헤드의 감소를 증명했다. 향후에는 송수신 데이터의 크기에 따라 동적으로 인터럽트 통합 수준을 결정함으로써, 송수신 인터럽트 오버헤드를 최적화 시킬 수 있는 비동기 시리얼 통신 모듈을 연구할 계획이다.

참고문헌

[1] 구철회, “디바이스 데이터 입출력에 있어서 폴링 방식과 인터럽트 구동 방식의 데이터 처리 방법”, 한국항공우주학회지, 제 33권, 9호, pp. 113-119, 2005. 9.

[2] 박석중, 우준, 이재국, 김형식, “과학계산용 클러스터 파일시스템에서의 인터럽트 통합효과 분석”, 한국 컴퓨터종합학술대회 논문집(D), 제 35권, 1호, pp. 105-109, 2008.

[3] Ravi Prasad, Manish Jain, Constantinos Dovrolis, "Dovrolis. Effects of interrupt coalescence on network measurements.", Passive and Active Measurements (PAM) conference, April 2004.

[4] Samsung Electronics, "S3C2440A 32-BIT CMOS MICROCONTROLLER USER'S MANUAL Revision 1", pp. 235-353, Samsung Electronics, 2004.
[5] Jean J. Labrosse, "MicroC/OS-II The Real-Time Kernel Second Edition의 역서", pp. 1-605, 에이콘, 2003.
[6] Jeffrey C. Mogul, K. K. Ramakrishnan, “Eliminating receive livelock in an interrupt-driven kernel”, ACM Transactions on Computer Systems (TOCS), v.15 n.3, pp. 217-252, 1997. 8.
[7] Jean J. Labrosse, “Embedded Systems Building Blocks Second Edtion의 역서”, pp. 399-494, 에이콘, 2008.
[8] 김희철, 이병수, “C언어와 통계학”, pp. 318-366, 상조사, 2000.
[9] (주)씨랩시스 역, “ARM System Developer's Guide”, pp. 455-747, 사이텍미디어, 2005.

오 삼 권(Sam-Kweon Oh) **[종신회원]**

- 1994년 : Queen's University (Canada)(컴퓨터과학박사)
- 1995년 3월 ~ 현재 : 호서대학교 컴퓨터공학부 교수

<관심분야>
Embedded Real-Time Systems, OS, Communication Protocol, Fault Tolerance

박 근 덕(Geun-Duk Park) **[정회원]**

- 2005년 8월 : 서울대학교 전기컴퓨터공학부 (공학박사)
- 2006년 3월 ~ 현재 : 호서대학교 컴퓨터공학부 조교수

<관심분야>
임베디드소프트웨어공학, 서비스 지향 컴퓨팅, XML 응용

김 병 국(Byoung-Kuk Kim) **[준회원]**

- 2009년 3월 ~ 현재 : 호서대학교 일반대학원 컴퓨터공학과 (컴퓨터공학 석사과정)

<관심분야>
Embedded System, RTOS