The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 16, No. 6, pp.109-117, Dec. 31, 2016. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2016.16.6.109
JIIBC 2016-6-13

RFID 시스템에서 충돌비트 위치를 이용한 M-ary QT 알고리즘 향상에 관한 연구

A study on enhanced M-ary QT algorithm using collision bits position in RFID system

김관웅*

Kwan-Woong Kim*

요 약 RFID 리더의 가장 중요한 기능은 자신의 인식 범위 내에 있는 개별 제품들에 부착된 태그의 고유한 EPC (Electronic Product Code) 코드를 식별하는 것이다. RFID 리더는 무선채널을 통하여 각각의 태그들에 쿼리 메시지를 전송하고, 태그들은 쿼리 메시지에 응답하여 동시에 자신의 식별코드를 전송하기 때문에 태그 충돌이 발생한다. 태그 충돌 방지 기법은 RFID 기술이 다양한 산업 분야에 적용되기 위해서 해결되어야하는 핵심적인 기술이다. 본 논문에서는 충돌비트의 위치 정보를 리더에서 뿐 아니라 태그에서도 이용할 수 있는 개선된 M-ary QT 알고리즘을 제안하였으며, 태그들로부터의 응답메시지를 분석하여 다수의 쿼리 메시지를 하나의 쿼리 메시지로 통합할 수 있는 기법을 제안하여 M-ary QT 알고리즘의 성능을 향상시켰다. 시뮬레이션 결과 제안된 알고리즘은 기존의 M-ary QT 기법에 비하여 질의-응답 수, 인식 효율, 통신비용 관점에서 매우 성능이 우수함을 보였다.

Abstract The most important mission of RFID reader is identify EPC (Electronic Product Code) of RFID tag of products that located within distinguishable range of RFID reader. RFID reader transmits query message to RFID tags through wireless channel and RFID tags send unique EPC to response its query message simultaneously. therefore tag collision occurred frequently. RFID tags collision resolution algorithm required to apply RFID technology to various industries.

In this paper, we propose enhanced M-ary algorithm that collision bits location is used by not only RFID reader but also tags. the main feature of the proposed algorithm is that integrate multiple query message of M-ary QT algorithm to the single query message by analyze multiple response messages from tags. the simulation results show that the proposed algorithm give better performance than M-ary QT algorithm in terms of the number of query-response, identification efficiency and communication overhead.

Key Words : RFID system, anti-collision, M-ary Query Tree, Manchester Coding

Ⅰ. 서 론

최근 유비쿼터스 환경을 완성하기위한 RFID/USN (Radio Frequency IDentification/Ubiquitous Senser Network) 기술은 RFID 칩의 기술이 소형화, 저가화 및 지능화됨에 따라 조달, 국방, 우편, 교육, 문화, 엔터테인

*정회원, (주)썬더테크놀로지
접수일자 : 2016년 8월 17일, 수정완료 : 2016년 10월 16일
게재확정일자 : 2016년 12월 9일

Received: 17 August, 2016 / Revised: 16 October, 2016 / Accepted: 9 December, 2016
*Corresponding Author: : kwkim@thunderspeaker.com
Thunder Technology LTD, Korea

먼트, 교통, 환경 등 다양한 분야에 적용되어 가고 있으며 결국 지능형 USN로 진화될 것이다. RFID는 각종 물품에 소형 칩인 태그(Tag)를 부착하고 사물의 정보와 주변 환경 정보를 판독·해독기능이 있는 판독기(Reader)를 통해 인식하여 무선주파수로 전송·처리하는 비접촉식 인식 시스템이다. RFID는 높은 인식률, 비접촉형 인식매체, 도달거리, 다른 통신망과의 연계 및 통신 가능성 등의 확장성으로 인해 항만/물류/유통, 군사, 식품/안전 등 비즈니스 영역에 킬러 애플리케이션으로서 막대한 파급 효과를 끼칠 전망이다[1, 16].

RFID 리더의 가장 중요한 기능은 자신의 인식 범위 내에 있는 개별 제품들에 부착된 태그의 고유한 EPC (Electronic Product Code) 코드를 식별하는 것이다. 식별된 제품의 EPC 코드는 해당 제품의 생산 정보나 유통 이력 등의 상세 정보를 사용자에게 제공한다.

리더는 무선채널을 통하여 인식범위 내의 태그에 태그 식별을 위한 요청 메시지를 전송한다. 리더로부터 요청 메시지를 받은 태그들은 동시에 리더로 자신의 식별 코드를 전송하기 때문에 태그 충돌이 발생한다[2]. 이 때 리더가 동시에 응답한 여러 개의 태그를 식별해야하는 문제가 발생하는데 이를 해결하는 기술이 충돌방지 (Anti-collision) 알고리즘이다. 태그 충돌 방지 기법은 RFID 기술이 다양한 산업 분야에 적용되기 위해서 해결되어야하는 핵심적인 기술이다.

태그 충돌 방지 기법은 크게 확률적인 (probabilistic) 방법과 결정적인(deterministic) 방법의 2가지로 구분된다. 확률적인 방법에서는 알로하(Aloha) 기반 프로토콜[3]이 대표적인데, 알로하 기반 프로토콜에서는 시간을 슬롯(slot) 단위로 나누고 태그들이 임의의 슬롯을 선택하여 응답한다. 만약 해당 슬롯에서 응답하는 태그가 하나만 존재하여 충돌이 발생하지 않는다면, 리더가 태그의 정보를 인식할 수 있다. 그러나, 해당 슬롯에 다수의 태그가 응답하여 충돌이 발생하면 다음 차례의 슬롯에 응답하여야 한다. 이러한 확률적인 방법은 본질적으로 확률의 불확실성을 가지고 있기 때문에 리더의 전송범위 내에 있는 모든 태그들을 식별하는 것을 보장하지 못하는 단점이 있다.

트리 기반의 태그 충돌 방지 프로토콜은 이진 탐색 알고리즘과 흡사하다. 트리 기반의 충돌 방지 프로토콜은 트리를 구성하는 노드들이 각각 하나의 질의-응답 과정에 해당한다. 따라서 트리의 깊이(depth) 하나를 감소하면 이진 트리(binary tree)의 경우에는 전체 질의-응답 횟수를 절반 정도 줄이는 효과를 얻을 수 있고, M-ary 트리일 경우에는 1/M만큼 질의-응답 횟수를 줄일 수 있다. 대표적인 트리 기반의 태그 충돌 방지 프로토콜로는 쿼리 트리(Query Tree) 기법, 충돌 트리(Collision Tree), M-ary Query Tree 등이 있다[4-6].

본 논문에서는 M-ary 쿼리 트리 기법을 기반으로 태그 인식 깊이를 줄일 수 있는 기법을 제안한다. M-ary 쿼리 트리 기법은 연속적인 m-비트인식 방식과 매핑함수를 제공하는데, 제안된 기법에서는 멘체스터 코딩기법을 이용하여 연속하지 않은 m-비트를 인식할 수 있는 알고리즘을 제안한다. 맨체스터 코딩 기법을 이용하면 리더에서 다수의 태그로부터 응답 메시지의 충돌 비트의 위치를 계산할 수 있다. 이러한 충돌 비트 위치 정보는 다음 쿼리 메시지를 생성하는데 이용한다. 또한 연속하지 않은 m-비트 위치를 쿼리 메시지에 전달하기 위해서 맨체스터 코딩 기법을 이용함으로써 불필요한 질의-응답 회수를 효과적으로 줄 일 수 있기 때문에 트리 기반 프로토콜의 식별시간을 현저하게 줄일 수 있다. 제안된 기법은 컴퓨터 시뮬레이션을 이용하여 M-ary 기법보다 태그 식별 시간 및 통신 효율성에서 우수한 성능을 보임을 확인 할 수 있었다.

II. 관련 연구

1. 맨체스터 코딩

트리 기반의 프로토콜에서는 리더의 질의에 대응하여 특정 태그가 응답한다. 만약 리더의 질의에 대응하는 태그가 하나이면 충돌이 발생하지 않지만, 다수의 태그가 대응하여 응답하면 충돌이 발생하게 된다. 태그 응답에서 충돌을 판단하는 방법에는 충돌 여부만을 판단하는 충돌 감지(collision detecting) 기법과 개별 비트 단위로 충돌을 검출하는 충돌 추적(collision tracking) 기법이 있다[2].

충돌 추적 기법은 맨체스터 코딩 (Manchester code) 기법을 사용하면 구현이 가능하다. 또한 충돌 비트의 위치를 이용하면 트리기반의 프로토콜의 효율을 더욱 향상시킬 수 있다.

맨체스터 코딩에서 하나의 비트는 전압의 값이 아니라 전압의 전이로 표현된다. 즉, '0' 비트는 양의 전이로

표현되고, '1' 비트는 음의 전이로 표현된다. 만약 두 개 이상의 태그들이 동시에 리더의 쿼리에 응답하면 특정 비트 위치에서 양의 전이와 음의 전이는 서로 상쇄되어 맨체스터 코딩에서는 허용되지 않는 무전이 상태가 되어 어느 비트 위치에서 충돌이 발생했는지를 추적할 수 있다. 그림 1은 맨체스터 코드(Manchester code)를 사용하여 개별 비트 단위로 충돌을 검출하는 예를 보여준다. Tag A의 ID는 10101010이고, Tag B의 ID는 10110010일 때, 두 태그가 각각의 ID를 동시에 전송하면 bit 4와 bit 5 구간 내에서 양의 전이와 음의 전이가 서로 상쇄되어 무 전이를 발생시켜 충돌이 발생했는지에 대한 정보를 알 수 있다. 따라서 bit 4와 bit 5에서 충돌이 발생한 것으로 검출된다.

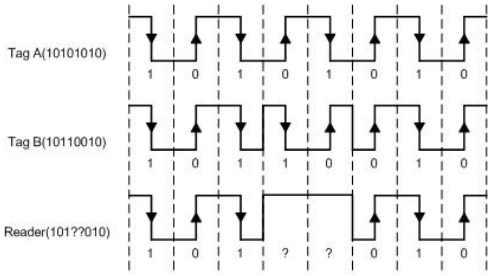

그림 1. 맨체스터 코딩의 예
Fig. 1. Collision Example of Manchester Coding

2. 쿼리 트리 (QT : Query Tree)

트리 기반의 프로토콜에서는 리더의 질의에 대응하여 특정 태그가 응답하고, 태그의 응답은 리더의 질의에 의해서만 결정되므로 메모리리스(memoryless) 방식이라고도 한다. 리더는 태그 ID의 일부(prefix)를 사용하여 질의하면 prefix를 수신한 태그 중에서 태그의 ID가 prefix와 일치하는 태그들만 응답한다. 이때 충돌이 발생하면 prefix의 길이를 1비트씩 늘려가면서 충돌을 일으킨 태그들을 두 부분 집합(subsets)으로 나누고 각각의 부분 집합에 대해 다시 쿼리를 보내서 식별하게 된다[5]. 그림 2는 4개의 태그(0000, 0101, 1100, 1110)를 식별과정을 보여준다. 초기에 리더가 'ε'(empty string)을 질의하면 4개의 모든 태그들이 응답하여 충돌이 발생한다. 초기 쿼리 'ε'에 의해 충돌이 발생하였음으로 리더는 prefix의 길이를 1비트 늘려 '0'과 '1'을 큐에 추가하고, 큐에서 '0'을 pop하여 다시 질의한다. 이 과정을 반복하여 리더가 '00', '01', '100', '111'의 prefix를 각각 질의할 때 마다 각각의 태그 ID가 식별된다. 한편 리더가 '10'의 prefix를 질문할 때 응답하는 태그가 없어 무응답 사이클이 발생하여 성능저하를 피할 수 없다. 그림 3과 같이 4bits 길이를 갖는 4개의 태그를 식별하는데 9번의 많은 질의-응답 과정이 필요하다.

이와 같은 쿼리 트리 기법은 리더의 전송범위 내에 존재하는 모든 태그들의 식별을 보장하지만, 한 번의 쿼리 전송에 하나의 비트만 식별할 수 있고, 충돌이 발생한 경우에 태그들의 정보를 확인할 수 없기 때문에 불필요한 질의-응답 과정인 무응답 노드가 추가되어 식별하는 시간이 상당히 길어질 수 있다.

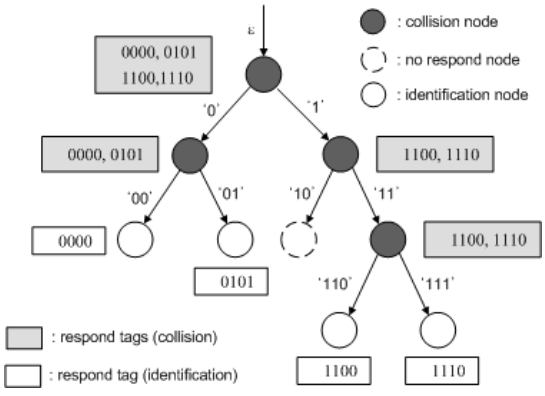

그림 2. 쿼리 트리 알고리즘
Fig. 2. Query tree algorithm

3. 충돌 트리 (CT : Collision Tree)

충돌 트리 알고리즘은 맨체스터 코드 (Manchester code)의 특성을 활용한 충돌 추적(collision tracking) 기법을 사용하여 QT 알고리즘의 성능을 개선하였다[7]. 리더는 충돌 추적(collision tracking) 기법으로 태그의 응답에서 충돌이 발생한 비트의 위치를 인식할 수 있으므로 충돌이 발생하기 이전의 비트 내용을 prefix 확장 시에 포함한다. 결국 리더는 충돌이 발생한 경우에만 prefix를 확장하여 재질의 하는데, 이러한 기법을 이용하면 무응답 노드가 발생하지 않아 QT 알고리즘의 지연시간을 획기적으로 줄일 수 있다.

4. M-ary Query Tree

QT는 하나의 질의-응답 과정에 하나의 비트만 식별하는 1-bit 인식 대신에 M-ary Query Tree 기법에서는 맨체스터 코딩 기법과 매핑함수를 이용하여 m-bits 인식

방법을 제공한다. 매핑함수는 m-bits 스트링을 M(2m)비트 스트림으로 변환하는 함수로서 표 1과 같은 변환 규칙을 이용하여 변환한다[9].

표 1. 2-bits 매핑
Table 1. Mapping Table of 2-bits

2 bits	$4(2^2)$ bits
00	0001
01	0010
10	0100
11	1000

그림 3은 4개의 태그(00001011, 00110110, 10011000, 11001101)를 식별하는 2-ary QT 알고리즘의 예를 보여준다. 초기에 리더가 'ε'(empty string)을 질의하면 4개의 모든 태그들이 응답하는데, 태그 아이디의 최상위 2bits를 매핑함수를 이용하여 변환하고 나머지 아이디 비트들을 추가하여 맨체스터 코드로 변환하여 응답한다.

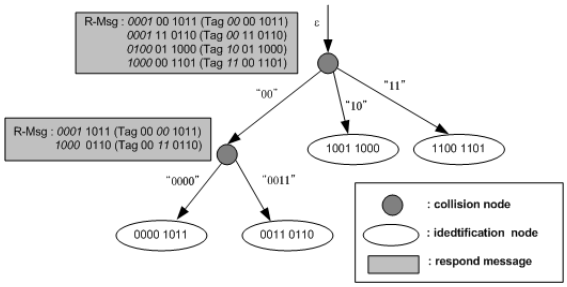

그림 3. M-ary QT 알고리즘
Fig. 3. M-ary Query tree algorithm

응답 메시지를 수신한 리더는 매핑함수를 이용하여 응답 메시지의 비트들 중에서 최상위로부터 M비트의 매핑비트를 역변환하여 태그들의 ID 정보들을 추출하여 실제로 존재하는 태그들에 대한 질의들만을 생성하여 질의-큐에 추가한 다음 질의-응답 과정을 반복한다.

MQT는 유휴노드의 개수를 없애며, m-비트인식 방식을 이용하여 충돌 노드의 개수를 상당히 줄일 수 있다.

Ⅲ. 제안된 M-ary QT 알고리즘

제안된 기법은 M-ary Query Tree 기법을 수정하여 쿼리 메시지의 임의의 위치에 비연속적인 m-bits의 인식 비트를 전송하고, 쿼리에 해당하는 태그에서 매핑함수를 이용하여 M(2^m) 비트 스트림으로 변환하여 응답하는 알고리즘을 제안하였다. 위와 같이 임의의 위치 m-bits 질의-응답 과정은 연속적이지 않은 충돌비트 위치를 연속적인 충돌 비트 인식 방식과 같은 효율성으로 인식할 수 있기 때문에 M-ary QT 알고리즘의 성능을 상당히 개선할 수 있다.

리더는 prefix에 m-bits이하의 '*' 인식비트가 포함되는 쿼리 메시지를 생성하여 맨체스터 코딩 기법으로 인코딩하여 전송한다. 쿼리 메시지는 맨체스터 코딩 기법을 이용하여 '0' 비트는 양의 전이로 표현되고, '1' 비트는 음의 전이로 표현되고, '*' 인식비트는 전이가 없는 충돌 비트 패턴으로 인코딩 되어 전송된다. 쿼리 메시지를 수신한 태그들은 그림 4와 같이 충돌 비트를 제외한 prefix와 태그 ID가 일치하면 충돌 비트 위치와 태그 ID로부터 인식비트 m-bits를 계산하여, 매핑함수를 이용하여 m-bits를 M bits로 확장하고 나머지 태그 ID를 추가하여 응답 메시지를 생성한다.

```
Processing incoming query messages in Tag
input : QM[qsize];  // received query message
        TID[isize];  // Tag id
output : mbits[msize]; // m-bits recognition
         RM; // respond message
for (i = 0; i < qsize; i++) {
    if (QM[i] == '*')  mbits[maddr++] = TID[i];
    else if (QM[i] != TID[i])
            NO Respond Message And End Processing;
}
if (maddr < msize) {
    for (j = 0; j < msize - maddr; j++)
        mbits[maddr++] = TID[qsize + j];
}
RM = get M(2m) bits stream with mbits[msize];
RM = RM + the remaining bits of the tag ID;
Send the respond message (RM)
```

그림 4. 태그에서 수신 쿼리 메시지 처리
Fig. 4. Processing incoming query messages in Tag

생성된 응답메시지는 맨체스터 코딩 기법으로 인코딩하여 리더에 전송한다. 응답 메시지를 수신한 리더는 응답 메시지를 분석하여 태그를 인식하거나 다음 쿼리 메시지를 생성하여 쿼리 큐에 추가하고 큐에서 다음의 쿼리 메시지를 pop하여 전송하는 과정을 모든 태그를 인식할 때까지 반복한다. 리더에서 수신 메시지를 처리하는 과정은 다

음과 같다. 수신 메시지는 M bits의 확장 비트 영역과 태그의 나머지 ID 부분으로 구성되어 있다. 리더는 우선적으로 그림 5와 같은 기법으로 확장 비트 영역의 모든 비트 '1'과 충돌 비트 '*'의 위치정보와 표 1과 같은 매핑함수의 역변환으로 m-bits 인식비트 리스트를 구한다.

```
Processing incoming query messages in Reader
input : RM[Msize];       // M bits of receive messages
output : mbitsList;          // m-bits recognition list
for (i = 0; i < Msize; i++) {
    for (j = 0; j < Msize; j++) M[j] = '0';
    if (RM[i] == '1' or RM[i] == '*')  {
        M[i] = '1';
        rembits = get reverse mapping function with M;
        mbitsList.add (rembits);
    }
}
```

그림 5. 수신 메시지의 M bits 확장 영역 처리
Fig. 5. Processing M bits of receive messages

```
Algorithm for prefix lists with mbits list in Reader
input : mbitsList;       // m-bits recognition list
            QM[qsize];    // send query message in reader
output : prefixList;  // prefix list for Tag identification
for (i = 0; i < qsize; i++)
    if (QM[i] == '*') cnum++; // get collision bits num
if (cnum < msize)
    for (i=0; i < msize  -  cnum; i++)
        QM[qsize + i] = '*';                    // add collision bits
for each (mbits in mbitsList) {
    maddr = 0;
    for (i = 0; i < QM.size; i++)
        if (QM[i] == '*') prefix[i] = mbits[maddr++];
        else prefix[i] = QM[i];
    prefixList.add (prefix);
}
```

그림 6. prefix lists를 구하기 위한 Algorithm
Fig. 6. Algorithm for prefix list with mbits list

다음으로 이렇게 구한 m-bits 인식비트 리스트와 전송한 쿼리 메시지를 이용하여 그림 6과 같은 알고리즘으로 Tag 인식을 위한 prefix 리스트를 구한다. prefix 리스트는 응답 메시지의 충돌 여부에 따라 Tag를 인식하거나 새로운 쿼리 메시지를 생성하기 위해 사용된다.

prefix 리스트는 응답 메시지의 충돌비트 '*' 발생 여부에 따라 Tag 인식이나 새로운 쿼리 메시지를 생성하기 위해 사용된다. 태그를 인식할 수 있는 경우는 다음의 두 가지 이다. 첫 번째 조건은 M bits 확장영역을 제외하고 태그의 나머지 ID 부분에서 충돌이 발생하지 않은 경우이고, 인식되는 태그 아이디는 각각의 prefix list 에 태그의 나머지 ID 부분을 추가하여 인식된다. 따라서 첫 번째 조건에서는 prefix list 개 수 만큼 태그를 인식할 수 있다. 두 번째 조건은 M bits 확장영역에 하나의 비트 '1'과 태그의 나머지 ID 부분에서 하나의 충돌 비트 '*'가 발생한 경우이다. 위와 같은 조건이 발생할 경우 인식되는 태그의 수는 2개 이다. 왜냐하면 prefix 리스트에는 오직 하나의 prefix 가 들어있는 경우이고 태그의 나머지 부분에서 오직 한 개의 비트만이 충돌이 발생했기 때문에 prefix에 태그의 나머지 부분을 추가하고 충돌비트 부분을 비트 '0'과 비트 '1'로 대치하면 두 개의 태그를 인식할 수 있다. 태그를 인식할 수 있는 두 가지 경우를 제외하고는 새로운 쿼리 메시지를 생성하게 된다. 새로운 쿼리 메시지를 생성하는 알고리즘은 다음의 두 가지 경우로 나누어진다. 첫 번째의 경우는 태그의 나머지 부분의 충돌비트 '*' 수가 m 보다 크거나 같은 경우이고 그림 7과 같은 알고리즘으로 새로운 쿼리 메시지를 생성하여 쿼리 큐에 저장한다. 두 번째의 경우는 태그의 나머지 부분의 충돌비트 '*' 수가 m 보다 작을 경우이다. 이러한 경우에 prefix list 의 모든 prefix list 들의 충돌 비트 '*'수와 나머지 부분의 충돌비트 '*' 수를 더하여 m 보다 큰 경우에는 그림 7과 같은 알고리즘으로 새로운 쿼리 메시지를 생성하여 쿼리 큐에 저장한다. 그런데, prefix list 충돌비트 수와 태그의 나머지 부분의 충돌비트 수가 m 보다 작거나 같은 경우에는 그림 8과 같이 여러 개의 쿼리 메시지를 하나의 쿼리 메시지로 통합할 수 있어서 태그 인식을 위한 쿼리-응답 과정을 줄일 수 있어 효율을 높일 수 있다.

```
Algorithm for new query message (cnum >= m)
input : prefixList;  // prefix list for Tag identification
    RMTID[rsize]; // The remaining part of the tag's ID
output : newQM; // new query message
for each (prefix in prefixList) {
    for (i = 0; i < prefix.size; i++)
        newQM[qaddr++] = prefix[i];
    for (i = 0; i < rsize; i++) {
        newQM[qaddr++] = RMTID[i];
        if (RMTID[i] == '*')
            if (++num == m)  break;
    }
    for (i = newQM.size - 1; i >= 0; i--) {
        if (newQM[i] == '*') remove newQM[i] bit '*';
        else break;
    }
    add query queue (newQM);
}
```

그림 7. 새로운 쿼리 메시지 생성 알고리즘 (cnum >= m)
Fig. 7. Algorithm for new query message (cnum >= m)

그림 9는 제안된 알고리즘의 Tag 인식 과정을 보여주고 있다. 리더는 start query message 'ε'를 전송하여 태그 인식과정을 시작하고 수신 메시지를 분석하여 태그를 인식하거나 다음에 전송할 쿼리 메시지를 생성하여 쿼리 큐에 저장하고 하나의 쿼리 메시지를 pop하여 전송하고 수신 메시지를 분석하는 과정을 모든 태그를 인식할 때까지 반복한다. 그림 9와 같은 예에서 쿼리 메시지 (2) '0010*1'와 응답 메시지 (2) '*000*0* 00'의 경우를 보면 응답 메시지의 8 bits 확장영역 '*0 00*0*0'에서 인식 비트는 '001', '011', '111' 임을 알 수 있고, 확장영역 이외에 비트 영역 '0'에서 충돌이 발생하지 않아 쿼리 메시지 (2)에 인식비트를 대입하고 마지막에 태그의 나머지 비트 '0'을 추가하면 다음의 태그 ID들을 '001001010', '001001110', 001011110'를 인식할 수 있다.

```
Algorithm for new query message (cnum < m)
input : prefixList;  // prefix list for Tag identification
    RMTID[rsize]; // The remaining part of the tag's ID
output : newQM; // new query message
newQM = a prefix in prefixList;
for (i = 0; i < prefix.size; i++) {
    bit0 = 0; bit1 = 0;
    for each (prefix in prefixList) {
        if (prefix[i] == '0') bit0 = 1;
        if (prefix[i] == '1') bit1 = 1;
        if (bit0 == 1 and bit1 == 1) {
            cnum++;
            newQM[i] = '*';
            break;
    }
}
for (i = 0; i <  rsize; i++) {
    if (RMTID[i] == '*') {
        cnum++;
        newQM[prefix.size + i] = '*';
    } else newQM[prefix.size + i] = RMTID[i];
}
if (cnum > m)
    go algorithm for new query message (cnum >= m);
else
    add query queue (newQM);
```

그림 8. 새로운 쿼리 메시지 생성 알고리즘 (cnum < m)
Fig. 8. Algorithm for new query message (cnum < m)

쿼리 메시지 (5) '1010*1'와 응답 메시지 (5) '00000001*'의 경우에서는 확장영역 '00000001'에서 '000' 임을 알 수 있고, 태그의 나머지 비트의 영역에서 오직 하나의 충돌 비트만이 발생한 경우 이므로 '000' 비트를 쿼리 메시지 (5)에 대입하면 '10100100' 이고 나머지 한 비트에서 충돌이 발생한 경우이기 때문에 '101001000'과 '10100101'의 태그를 인식할 수 있다. 특히 쿼리 메시지 (3) '0110*1'과 응답 메시지 (3) '000*0*0**'을 살펴보면 화장영역 '000*0*0*' 에 의해서 인식 비트 들은 '000', '010', '100' 임을 알 수 있다.

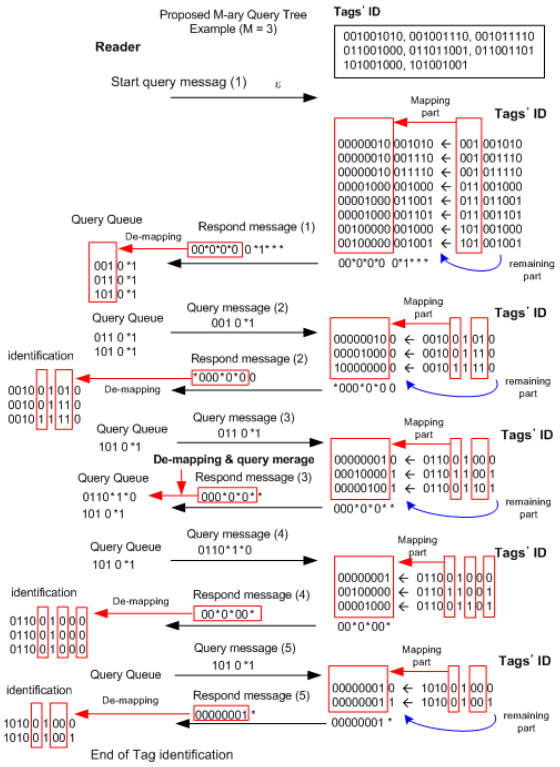

그림 9. 제안된 태그 인식 기법의 동작 예 (m=3)
Fig. 9. Example of proposed Tag identification

인식 비트를 쿼리 메시지 (3)에 대입하여 다음 쿼리 메시지를 생성하면 '01100100*', '01100110*', '001011 00*'와 같은 3개의 쿼리 메시지가 생성되는데, <그림 8> 과 같은 알고리즘을 적용하면 '000', '010', '100'에서 2개의 충돌 비트 '**0'와 태그의 나머지 비트에서 하나의 충돌 비트 '*'가 발생하여 3개의 충돌 비트가 되므로 '0110*1*0'와 같은 하나의 쿼리 메시지를 생성할 수 있다. 따라서 제한된 쿼리 메시지의 통합 기법은 쿼리 응답 싸이클 수를 효과적으로 줄일 수 있는 장점이 있다.

Ⅳ. 실험 및 결과

제안된 알고리즘의 성능평가를 위해서 M-ary 쿼리

트리 알고리즘과 제안된 알고리즘의 프로토콜을 시뮬레이션 프로그램으로 작성하여 비교 분석하였다. 실험 환경은 하나의 리더와 다수의 태그들로 구성되어 있다. 모든 태그는 리더의 인식 범위 내에 있으며, 태그 ID는 무작위로 부여된 64 bits 길이의 균일한 분포를 가지고 있다. 태그의 개수는 40에서 640까지 증가시키면서 10회 반복 실험한 결과의 평균값을 가지고 성능을 비교하였다.

시뮬레이션 결과는 다음의 세 가지 평가 요소를 기준으로 비교하였다. 첫 번째로 모든 태그를 인식하기 위한 쿼리-응답 사이클의 수로 태그 인식 시간과 관련이 있으며 가능한 적은 수의 사이클로 태그를 인식해야 한다. 두 번째로 태그의 수를 질의-응답 횟수로 나누어 인식효율을 계산하였다. 즉, 인식효율 평가요소는 한번의 질의-응답 과정에서 평균 몇 개의 태그를 인식할 수 있는지를 평가할 수 있다. 세 번째는 하나의 태그를 인식하는데 전송되는 평균 비트수를 계산하였으며, 이러한 평가요소는 에너지 소모와 관련이 있는 통신비용을 평가 할 수 있다.

그림 10은 4-ary QT 알고리즘과 제안된 4-ary QT 알고리즘의 전체 질의-응답 사이클 수를 보여주고 있다. 제안된 알고리즘의 질의-응답 회수는 기존의 4-ary QT 알고리즘에 비하여 태그의 수에 관계없이 평균 43% 이상 줄어든 결과를 보여주고 있다. 그림 11과 그림 12는 태그 인식 효율과 통신비용의 결과를 보여주고 있으며, 제안된 알고리즘은 기존의 4-ary QT 알고리즘에 비하여 인식 효율은 약 1.7배 높고, 통신비용은 0.8배 낮은 결과를 보여주고 있다. 이러한 결과는 제안된 알고리즘이 충돌 비트 위치를 효율적으로 이용하여 불연속 비트를 인식할 수 있을 뿐만 아니라, 조건에 따라 다수의 쿼리 메시지를 하나의 쿼리 메시지로 통합하는 등의 매우 효율적인 태그 인식 기법을 적용하고 있기 때문으로 생각된다.

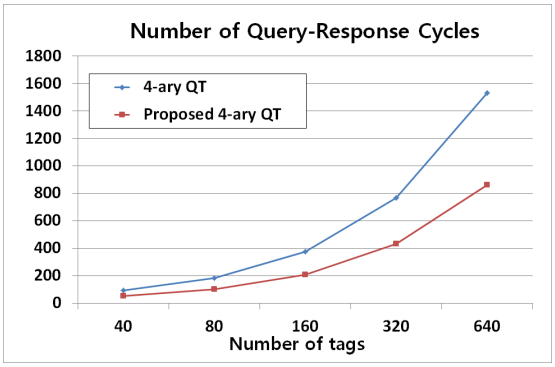

그림 10. 질의-응답 수
Fig. 10. Number of query-response cycles

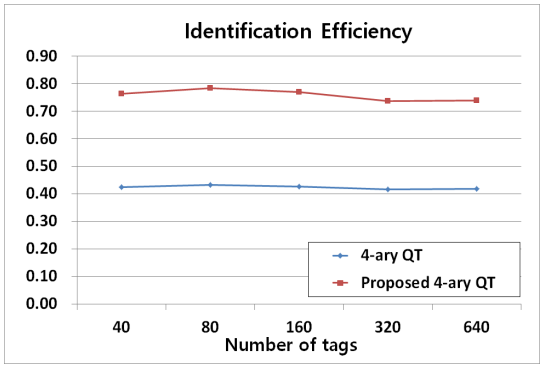

그림 11. 인식 효율성
Fig. 11. Identification Efficiency

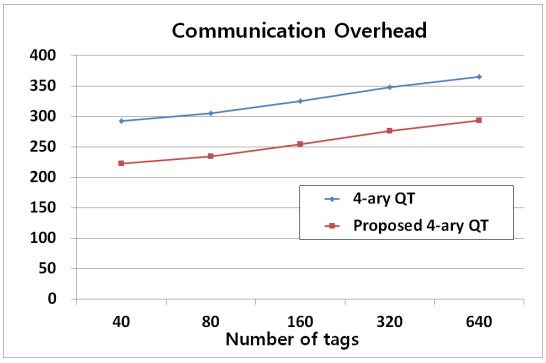

그림 12. 통신 비용
Fig. 12. Communication Overhead

V. 결 론

RFID는 IC칩과 무선을 통해 자동으로 물체에 부착 된 태그를 식별하고 태그에서 센싱 정보나 이력정보를 전송하여 식품, 동물, 사물 등 다양한 개체의 정보를 관리할 수 있는 차세대 인식기술이다. 다양한 산업 분야에 RFID 기술을 이용하기 위해서 많은 기술이 개발되고 있으며, 그 중에서 하나의 중요한 핵심 기술은 다수의 태그를 인식하는 기술이다. 모든 태그를 인식할 수 있는 결정적인 (deterministic) 방법 중에서 맨체스터 코딩 기법을 기반으로 하는 M- ary QT 기법은 많은 수의 태그를 인식하기 위한 다양한 분야에 적용되고 있는 기법이다.

본 논문에서는 충돌비트의 위치를 정보를 리더에서 뿐 아니라 태그에서도 이용할 수 있는 개선된 M-ary QT 알고리즘을 제안하였다. 리더에서 태그로 전송하는 쿼리

메시지에 충돌비트를 추가함으로써 비연속적인 m bits 인식이 가능하였다. 또한 태그들로부터의 응답메시지를 분석하여 다수의 쿼리 메시지를 하나의 쿼리 메시지로 통합할 수 있는 기법을 제안하여 M-ary QT 기법의 효율을 향상시킬 수 있었다.

시뮬레이션 결과 제안된 알고리즘은 기존의 M-ary QT 기법에 비하여 질의-응답 수는 43% 이상, 인식 효율은 약 1.7배, 통신비용은 0.8배 향상된 결과를 보여주고 있다. 시뮬레이션 결과에서 보여주듯이 제안된 M-ary QT 알고리즘은 충돌비트 위치 정보를 효율적으로 이용하여 태그 인식과정의 시간을 효과적으로 줄일 수 있음을 알 수 있다. 추후 연구과제로 태그 인식 과정의 효율을 향상시키기 위해 M-ary QT 알고리즘의 인식 비트 수를 가변적으로 적용할 수 있는 기법을 연구하고 있다.

References

[1] K. Finkenzeller, “RFID Handbook:Fundamentals and Applications in Contactless Smart Cards and Identification”, John Wiley & Sons, 2003.

[2] C.H. Quan, W.K. Hong, Y.D. Lee, H.C. kim, “Performance Evaluation of Anti-collision Algorithms in the Low-cost RFID System,” Journal of KICS, Vol. 30, No. 1B, pp.17-26, 2005.

[3] M. A. Bonuccelli, F. Lonetti, and F. Martelli, “Instant coll- ision resolution for tag identification in RFID networks,” Ad Hoc Networks, vol. 5, no. 8, pp. 1220-1232, Nov. 2007.

[4] J. Myung, W. Lee, J. Srivastava, and T. K. Shih, “Tag-splitting: adaptive collision arbitration protocols for RFID tag identification,” IEEE Trans. Parallel Distributed Syst., vol. 18, no. 6, pp. 763-775, June 2007.

[5] J. H. Choi, D. Lee, and H. Lee, “Query tree-based reservation for efficient RFID tag anti-collision,” IEEE Commun. Lett., vol. 11, no. 1, pp. 85-87, Jan. 2007.

[6] X. Jia, Q. Feng, and C. Ma, “An efficient anti-collision protocol for RFID tag identification,” IEEE Commun. Lett., vol. 14, no. 11, pp. 1014-1016, Nov. 2010.

[7] X. Jia, Q. Feng, C Ma, “An Efficient Anti-Collision Protocol for RFID Tag Identification,” IEEE Commun. Lett., Vol. 14, No. 11, pp.1014-1016, 2010.

[8] Feng Zhou, Dawei jin, Chenling Huang, Hao Min, “White Paper: optimize the power Consumption of Passive Electronic Tags for Anti-collision Schemes,” Auto-ID center Fudan Univ., October, 2003.

[9] J. Shin and D. Yang, “Multiple RFID tags identification with M-ary query tree scheme,” IEEE Commun. Lett., vol. 17, no. 3, pp. 604-607, Mar. 2013.

[10] G. Bagnato, G. Maselli, C. Etrioli, and C. Vicari, “Per- formance analysis of anti-collision protocols for RFID systems,” in Proc. IEEE 69th Veh. Technol. Conf., pp.1-5, Barcelona, Spain, Apr. 2009.

[11] D.M. Yang, J.M. Shin, “EMQT : A Study on Enhanced M-ary Query Tree Algorithm for Sequential Tag IDs”, The Korean Institute of Communications and Information Sciences, Vol.38B, No.06, pp. 435-445, June 2013.

[12] Y. Kim, S. Kim, S. Lee, and K Ahn, “Improved 4-ary query tree algorithm for anti-collision in RFID system,” in Proc. IEEE AINA 2009, Bradford, U.K., pp. 699-704, May 2009.

[13] EPC Tag Data Standards Version 1.1 Rev.1.24, [Online], Available: http://read.pudn.com/downloads46/sourcecode/others/155143/EPCTagDataSpecification11rev124.pdf, Apr. 2004.

[14] H. Gou and Y. Yoo, “Bit collision detection based query tree protocol for anti-collision in an RFID system,” Int. J. Innovative Comput., Inform. Control, vol. 8, no. 5, pp. 3081-3102, May 2012.

[15] 박춘명, “임베디드 멀티미디어 시스템에 기초한 글로벌유비쿼터스 시스템 구성 방안”, 한국인터넷방송통신TV학회 논문지 제6권 제2호, pp 29-34, 2006년 6월

저자 소개

김 관 웅(정회원)

- 1996년 전북대학교 전자공학과 졸업 (학사)
- 1998년 전북대학교 대학원 전자공학과(공학석사)
- 2002년 8월 전북대학교 대학원 전자공학과(공학박사)
- 2010년 ~현재 (주)썬더테크놀로지

<주관심분야 : 무선통신시스템, DSP 신호처리, 임베디드 시스템, QoS>