One Cambridge Center, Cambridge, MA 02142 • Tel: 866-833-4069 • www.thingmagic.com 8/11

Designing Bulk Encoding Applications Using UHF RFID Technology

By Michael Klein

Introduction

UHF passive RFID is becoming increasingly popular in the retail market, particularly for tagging apparel, due to its cost effectiveness, ease of use, scalability and reliability. To improve supply chain visibility, many retailers would like apparel suppliers to affix tags to items at the point of manufacture. This results in the need to encode the tagged items at a later point in time, often after they have been packaged.

This application note describes how one might design a robust bulk encoding system using ThingMagic UHF RFID technology. We assume the reader of this application note is generally familiar with both RFID technology and the ThingMagic MercuryAPI.

Note: *Code samples are provided as examples and will need to be modified for your solution environment.*

RFID Bulk Encoding

The process of bulk encoding UHF RFID tagged items has both hardware and software components. When bulk encoding a packaged set of tagged items, the first (and often most challenging) step is to identify all the tags, and only the desired tags, to be encoded. Once uniquely identified, the process of encoding the tags is relatively simple.

The following sections outline the software process for uniquely identifying the tags to be encoded, encoding them, detecting any missed writes, and handling missed writes. In addition, some of the specific hardware-related challenges will be identified and discussed.

ThingMagic RFID Readers

Variety of RFID readers –

ThingMagic offers a large selection of UHF RFID readers, all with state-of-the-art performance. ThingMagic RFID readers are available with USB, RS232, Ethernet and Wi-Fi interfaces, providing a variety of options to connect them to the rest of the system. Some ThingMagic RFID readers are designed for rugged environments, with some models providing up to 4 antenna ports and the option to use RF multiplexers for broader coverage. **The ThingMagic Mercury6e (M6e)**

reader module is ideal for high-speed bulk-encoding of tags. The M6e supports 4 antenna ports and will operate at RF output power levels of up to 31.5dBm from each port, delivering the flexibility to use multiple antennas to provide an optimal write zone. Its high maximum transmit level and low minimum receive level ensures successful reads and write, even when tags are unfavorably oriented. Further, it can be configured to operate at high rates of speed, reading up to 400 tags per second, by utilizing the fastest allowable Gen2 modulation scheme ("FM0" with a backscatter link frequency of 640KHz), increasing overall bulk-encoding throughput.

More info - http://www.thingmagic.com/embedded-rfid-readers/mercury6e

Embedded reader advantage –

The core of the ThingMagic technology lies in embedded modules that are integrated with other hardware to develop finished readers. These modules, employing the Atmel ARM microcontroller and a dedicated RFID ASIC, provide powerful RFID read and write capabilities in a small form factor. The modules are designed to be powered by a single +5VDC supply (+3 to 5.5VDC for M5e-Compact). A single cable provides power, data, and control to the modules, eliminating the complexity of multiple connection points and simplifying hardware integration. The same MercuryAPI, used for all ThingMagic readers, provides customers a broad range of configuration options and tuning capabilities to maximize performance for each specific use-case.

More info: http://www.thingmagic.com/embedded-rfid-readers

Seamless software integration –

The ThingMagic reader SDK is compatible with the following platforms – C, C#/.Net and Java. The SDK is designed for cross-product development allowing software engineers to create an interface to middleware for all ThingMagic readers, with no special installation of drivers and dlls required for specific readers. Most function calls and parameters are the same for all ThingMagic readers, with additional unique features available on particular readers or reader modules.

NOTE: *Antenna selection and placement are important elements of a successful RFID solution performance. To create a read zone which ensures precise and uninterrupted tag-reader communications, the RFID antennas you select must meet the performance specifications required by the environments and conditions in which you will be reading/writing tags. ThingMagic offers a variety of RFID antennas for solution development and can assist with selecting the appropriate antennas for your development and deployment needs.*

Configuration and algorithm guidelines with code samples

M6e Reader –

The sample code below shows how to configure the M6e reader to bulk-encode a box of tagged items with a list of provided EPC ID values.

Please note: the code below is a guideline and will need to be modified to work in your environment.

Step 1: Configure the reader –

Once a reader connection has been made, the following configuration should be performed:

Select the antennas to be used to write with. The reader can be configured to dynamically switch among all the antennas it detects, or a desired list of antennas can be "manually" specified.

```
int[] antennasToWriteWith =
   (int[])r.ParamGet("/reader/antenna/connectedPortList");
```

Unless all tags in the filed will have the same value written to them, it is necessary to identify individual tags based on a known unique data field. Typically that won't be the EPC ID field as they may all be the same (especially if they still contain their factory default value). In this example, we will use the TID memory bank's Unique Identifier (UID) field, although another data field may be used as long as it provides the required uniqueness. Most current-generation tags have a factory-programmed, read-only UID value. In order to configure the reader to use both the EPC ID and the chosen data field to identify an individual tag during the inventory process, the parameter "uniqueByData" must be set.

```
r.ParamSet("/reader/tagReadData/uniqueByData", true);
```

The following Gen2 parameter choices are recommended in order to maximize inventory performance. These settings minimize the number of repetitious tag responses and enable the highest data rate, resulting in both the highest read rate and greatest number of unique tag responses.

```
r.ParamSet("/reader/gen2/session", Gen2.Session.S1);
r.ParamSet("/reader/gen2/target", Gen2.Target.A);
r.ParamSet("/reader/gen2/BLF", Gen2.LinkFrequency.LINK640KHZ);
r.ParamSet("/reader/gen2/tagEncoding", Gen2.TagEncoding.FM0);
```

The "commandTimeout" parameter specifies how long to continue to retry a successful tag operation, in this case the write operation, before it gives up. Since different tags perform writes at different rates, this may need to be tuned, possibly for each tag model used. In general this setting should be configured to provide enough time to allow more than one write attempt on each tag. 100ms is a good setting for most tags. The tradeoff is essentially how long to retry writing a tag with a particular antenna versus giving up and trying on a different antenna.

```
r.ParamSet("/reader/commandTimeout", 100));
```

Step 2: Identify the tags to be written –

As mentioned in Step 1, the tags must be uniquely identifiable in order to "singulate" and write to them (unless all tags in the filed are to receive the same value). In this example, we will

assume that the portion of the TID UID field, which is stored in TID memory words 3, 4 and 5 (zero based) will be used for every tag. Note: not all tags have the same size TID memory bank, so these locations will vary by tag IC type. Also, not all tags which have UIDs have the same length UID, so it is important to know the RFID IC used by the tags being written.

This step is performed by setting up a `Gen2.ReadData TagOp` to read the data and setting it as an embedded `TagOp` of our `ReadPlan`. This causes the `ReadData` operation to be performed on every tag found during the inventory operation and included in the resulting `TagReadData` object created for each tag found.

```
TagOp readMemOp = new Gen2.ReadData(Gen2.Bank.TID, 3, 3);
SimpleReadPlan plan = new SimpleReadPlan(antennasToWriteWith,
   TagProtocol.GEN2, null, readMemOp, 100);
r.ParamSet("/reader/read/plan", plan);
```

***NOTE**: If all the tags to be written have known common data fields that can be used to distinguish them from other tags in the area (for example if tags in a box are pre-encoded with a common EPC or common data in User Memory), this can be used as a filter so that tags in other nearby boxes aren't included in the list of tags to write. This process can be of great benefit for combating one of the major challenge of bulk encoding at high speeds (see Challenges section) and would be done by creating a Select filter and specifying it in place of the 'null' in the SimpleReadPlan creation shown in the example above.*

Execute the read. The ideal timeout for the inventory (Read) operation depends on the specifics of the use case. After the Read operation, `tagReads` should contain all the tags to be written.

```
TagReadData[] tagReads = r.Read(500);

//Initialize array to store failed writes
TagReadData[] writeFailedTags = new TagReadData[tagReads.Length];
```

Step 3: Provide list of IDs to write and write to the tags –

We now have the reader configured for maximum write performance, a set of antennas to use for write operations, and have identified a list of tags to write. The next step is to get a list of new IDs to write to the tag - we will assume the desired tagIDs are generated through another method, `GetNewEPCIDs()` which is intended to be a placeholder for the business process that will supply the new EPCs for the list of tags passed to it - and write them.

```
Byte[][] newIDs = GetNewEPCIDs(tagReads);
```

If multiple antennas are specified, the full list of tags to write will be attempted on the first antenna. Then, only those that were unsuccessful will be retried on subsequent antennas in the list. This is where the tuning of the "/reader/commandTimeout" parameter comes into play. Once all antennas have been used, if any tags were still not successfully updated they will be in the `writeFailedTags` list.

```
for (int antennaIndex = 0; antennaIndex < antennasToWriteWith.Length;
   antennaIndex++)
{
   r.ParamSet("/reader/tagop/antenna", antennasToWriteWith[antennaIndex]);
   if (antennaIndex == 0)
   {
      newIDs = GetNewEPCIDs(tagReads);
      writeFailedTags = BulkWriteTagEPCs(tagReads, newIDs, membank,
         DataReadStartOffset, numDataWords);
   }
   else if (writeFailedTags.Length>0)
   {
      newIDs = GetNewEPCIDs(writeFailedTags);
      writeFailedTags = BulkWriteTagEPCs(writeFailedTags, newIDs, membank,
            DataReadStartOffset, numDataWords);
   }
   else { break; }
}
```

The `BulkWriteTagEPCs` *method used above is as follows: It accepts a list of* `tagsToWrite`*, a list of the* `newEpcIDs` *to be written and the singulation selection criteria that will be matched against the unique identifier data from each* `tagsToWrite`*. Upon completion, the method returns a list of tags,* `writeFailedTags`*, which were not successfully written.*

```
private TagReadData[] BulkWriteTagEPCs(TagReadData[] tagsToWrite, byte[][]
   newEpcIDs, Gen2.Bank selectMemBank, uint selectDataReadStartOffset, ushort
   selectNumDataWords)
{
      int tagArrayIndex = 0;
      List<TagReadData> writeFailedTags = new List<TagReadData>();
      foreach (TagReadData tr in tagsToWrite)
      {
            try
            {
                  Gen2.Select selectOnData = new Gen2.Select(false,
                     selectMemBank, selectDataReadStartOffset * 16,
                     (ushort)(selectNumDataWords * 16), tr.Data);
                  ushort[] epcwords =
                     ByteConv.ToU16s(newEpcIDs[tagArrayIndex]);
                  r.ExecuteTagOp(new Gen2.WriteTag(new
                     Gen2.TagData(newEpcIDs[tagArrayIndex])), selectOnData);
             }
             catch (Exception ex)
            {
                  writeFailedTags.Add(tr);
             }
             tagArrayIndex++;
       }
       return writeFailedTags.ToArray();
}
```

Step 4: Handle missed writes –

Once the write operation has attempted writes using all antennas, tags which remain unwritten will likely need to be handled through a special process. At this point the specifics depend on the end-user requirements. Typical processes often include:

- Marking the box, indicating the number of unwritten tags. This assumes some percentage of unwritten tags is acceptable during the main bulk encoding process. They either get fixed at a future step in the process, likely using one of the following "exception handling" methods, or are simply left as acceptable failures.
- Manual "Exception handling" of the box – the box is pulled from the conveyor and someone manually finds and writes the unwritten tag(s). This requires additional custom code, primarily to find the tag and determine the EPC ID to be written. The writing process is straightforward.
- Automated "Exception handling" of the box – It is possible to automate the finding and writing of the unwritten tag(s). Since the written EPCs are known, it's simply a matter of inventorying the box, finding the exceptions and writing them. The logic is similar to the main process. The main challenge of including this logic in the main bulk encoding process is the additional time required to do it. It's best to divert exception boxes and handle them independently.

Challenges

The information above provides guidelines for the process, reader configuration and algorithm to enable a typical bulk encoding application. However, this alone does not necessarily solve all the requirements for a robust bulk encoding solution. One of the major challenges with bulk encoding is making sure you've identified all the tags to write and only write to those tags. More specifically there are two areas of concern:

1. *Reading all the tags in a box* – There are many environmental factors that can degrade the ability to read the UHF RFID tag on items. Conditions such as: (1) metal (buttons on pants, metallic decals, etc.) or liquids near a tag; and (2) poor orientation of the tag to the antenna can cause it to be difficult to read. If a tag isn't read during the inventory of the box, there is no way to know that it wasn't successfully written. One way to overcome this problem is to label the box, identifying the number of tagged items that are in it. If the inventory returns fewer or more items, it can be identified as a potential exception case and handled appropriately (see exception handling options above).
2. *Writes to tags in adjacent boxes* – This is probably the most challenging problem in relation to overall writing success. The easy/obvious solution is to move the boxes through an "RF tunnel" in which the RF is field is well contained and only tags inside the tunnel are readable and writable. At slow conveyor speeds this works well. However, at higher conveyor speeds, the tunnel may need to be lengthened to achieve the required time for uninterrupted writing, necessitating increased spacing between boxes, potentially negating the benefits of the higher conveyor speed. Unique approaches to these challenges are where solution differentiation can be provided.

Summary

UHF RFID provides tremendous benefit to all types of asset tracking applications, with retail apparel representing one of the fastest growing uses. Bulk encoding can provide additional gains in ROI by allowing manufacturers to delay customer-specific encoding criteria until the end of the packaging process, when it can be done “on-demand”.

This application note described the basic techniques used to identify and encode RFID tags using the ThingMagic RFID Readers and the MercuryAPI, with a focus on the retail use-case of bulk encoding. In addition to the retail-focused example described here, bulk encoding techniques can be used for any application where groups of tagged items need to be encoded. The simplicity, flexibility and speed of ThingMagic’s bulk encoding capabilities help make deployment and use of RFID easier than ever.

For more information, visit www.thingmagic.com.