One Cambridge Center, Cambridge, MA 02142 • Tel: 866-833-4069 • www.thingmagic.com 6/11

Designing Race Timing Applications Using UHF RFID Technology

By Kapil Asher

Introduction

UHF passive RFID is becoming extremely popular in race timing application development due to its cost effectiveness, ease of use, scalability and reliability. This application note describes how one might design a robust race timing system using ThingMagic UHF RFID technology. The application note assumes the reader to be generally familiar with RFID technology and the ThingMagic MercuryAPI.

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

UHF RFID in Race Timing

Typical race timing applications are comprised of three major parts -

Checking-in participants before the race –
Historically, race coordinators have set up registration booths to manually cross reference names on a sheet of paper to check-in racers on race day. This slow process can be eliminated by mailing pre-associated RFID-enabled race bibs or other race tags to the racers in advance. "RIFD enabled" participants get checked-in automatically via an RFID reader at the starting point. If race preparation beforehand is not possible, tags can be easily associated to participants on race day using desktop RFID readers.

Recording critical timestamps of a participant during the race –
Race coordinators record the time taken by each participant to complete a number of milestones. When handled manually, the process is tedious and prone to human error, especially in large races like marathons which are typically attended by thousands of runners. This timestamp data then gets fed to a database for results reporting, which can be further delayed if done manually. RFID systems can automate the collection of timestamps by reading the participant's tag at critical locations and updating them to a central database using GPRS/EDGE/3G technology or by utilizing the data write feature to store the timestamps and checkpoints directly on the tag, which is then interrogated at the end of the race for stored

information. This time-sensitive application requires fast data transfer between the reader and the tag and minimal back-end overhead.

Presenting data in a consumable format –
Race statistics are stored on servers that are accessible via web interfaces for participants to check their performance. RFID readers that are network-ready can easily stream data to servers via ethernet, WiFi, EDGE/GPRS/3G. In case connectivity cannot be achieved, the data can be stored on-board the RFID reader and transferred later.

ThingMagic UHF RFID Readers

Variety of RFID readers –

ThingMagic designs one of the largest varieties of UHF RFID readers in the market, with remarkable consistency in superior performance. ThingMagic RFID readers can be interfaced via USB, RS232, Ethernet and WiFi, giving expansive options to connect to the rest of the system. 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 USB RFID reader is designed for desktop applications and is ideal for associating tags to racers on race day. It is powered via USB and has an integrated antenna with a read range of less than 12 inches to ensure programming only at a close range, thus avoiding association to an unintended tag. The reader has 2 LEDs for visual confirmation of association success and 2 buttons for triggering events.

More info - http://www.thingmagic.com/usb-rfid-reader

The ThingMagic Mercury6 (M6) reader is a network-ready UHF RFID reader that is ideal for reading tags on participants on race day. Out-of-the-box, each M6 reader can be connected to 4 antennas, giving broad RF coverage on the racetrack or route. Its continuous mode of transmission allows no down time in polling tags which ensures no missed reads during the race. This mode is asynchronous to other processes in the system allowing uploading data to servers at the same time as reading new RFID tags. Its high sensitivity and a read rate of greater than 400 tags/sec ensures reads in unfavorable tag orientations. M6 readers can update tag data on the fly without the requirement of stopping reads, allowing the storage of timestamps and checkpoints on the tag memory quickly and efficiently. It can be configured to operate at higher physical data rates by utilizing the Gen2 FM0 modulation scheme and a backscatter link frequency of 640KHz. The M6 reader can read IP-X tags that have smaller transaction durations than Gen2 and may be favorable for races.

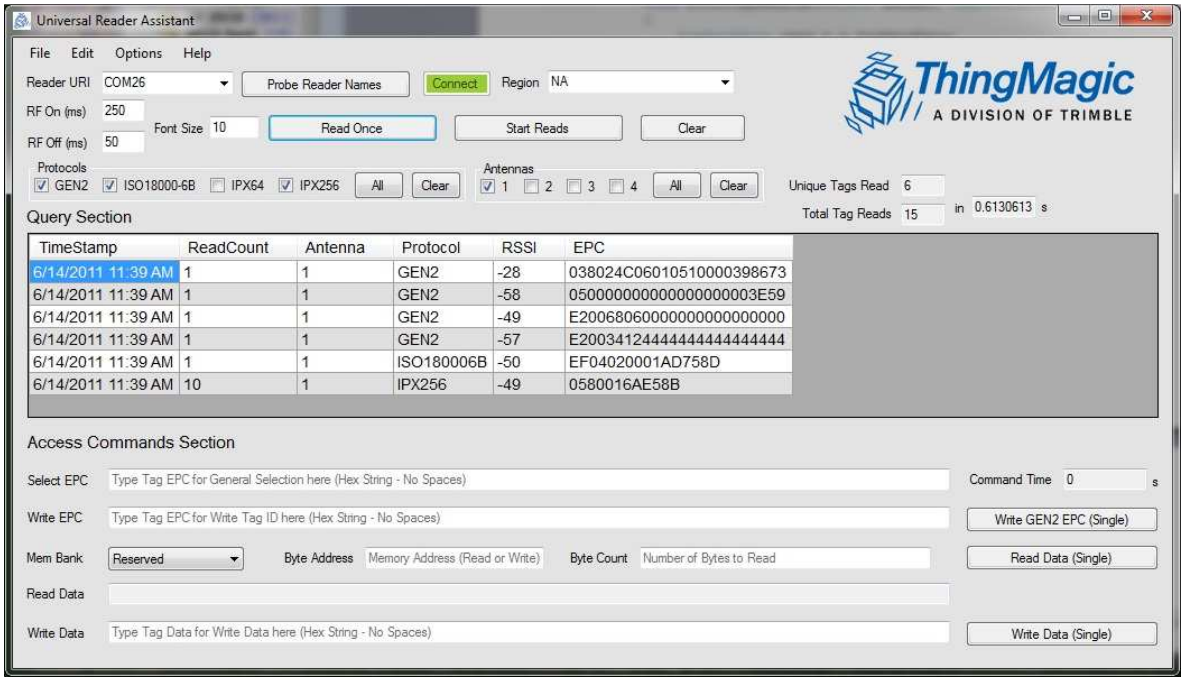
More info - http://www.thingmagic.com/fixed-rfid-readers/mercury6

Seamless software integration –

ThingMagic readers are available with an SDK for the following platforms – C, C#/.Net and Java. The SDK is designed for cross-product development allowing software engineers to program middleware for all ThingMagic readers without special installation of drivers and dlls required

for specific readers. Function calls and parameters are the same for all readers with the exception of unsupported features on particular reader hardware. An example of a middleware that works with any ThingMagic reader can be found here (Universal Reader Assistant) - http://www.thingmagic.com/support-login?username=api&return=/download?func=startdown%26id=36

The ThingMagic Universal Reader Assistant (screen shot below) is one of several tools provided by ThingMagic to help with developing powerful RFID-enabled solutions.



Embedded reader advantage –

The core of the ThingMagic technology lies in embedded UHF RFID modules that are integrated with other hardware to develop the finished readers. These modules, built over the Atmel ARM microcontroller and RFID ASIC are powered with RFID read and write capabilities in a small form factor. The modules are designed to work for +5VDC (+3 to 5.5VDC for M5eC). A single off-the-shelf cable provides power and data to the modules, eliminating complex solder jobs for hardware integration. ThingMagic modules can be seamlessly integrated with existing telematics hardware allowing faster RFID data transfer to a previously designed system. A high level of control is available via the same MercuryAPI used for ThingMagic finished readers, allowing customers to program their power consumption scheme, boot-up configuration scripts and data transfer rate.

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

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

Configuration guidelines with code sample

USB Reader –

A simple codelet below shows how to configure the USB reader to perform an association between the EPC tag and the race participant.

Please note: this codelet is a guideline and will need to be modified to work in your environment.

Step 1: Configure the reader –

```
private Reader initilizeSerialReaders(string comPort)
{
    Reader rdr1 = null;
    try
    {
        //Define a new reader object
        rdr1 = new SerialReader(string.Concat("/", comPort));
        rdr1.Connect();
        //Set a timeout the host waits for transport messages to and from the reader.
        //Useful for slow processors.
        rdr1.ParamSet("/reader/transportTimeout", int.Parse(asynchOnTime.Text) + 5000);
        //Set the legal region allowed for the reader in a given area
        Reader.Region regionToSet = Reader.Region.NA;
        rdr1.ParamSet("/reader/region/id", regionToSet);
        //Set Gen2 session to 0 as there is only 1 tag in the field and continuous response from the tag
        //favors faster encoding
        rdr1.ParamSet("/reader/gen2/Session", Gen2.Session.S0);
        //Set a high baud rate
        rdr1.ParamSet("/reader/baudRate", 115200);
        return rdr1;
    }
    catch (System.IO.IOException)
    {
        MessageBox.Show("Reader not connected on " + comPort, "Error!", MessageBoxButtons.OK);
        return null;
    }
    catch (ReaderCodeException ex)
    {
        MessageBox.Show("Error connecting to Reader: " + ex.Message.ToString(), "Error!", MessageBoxButtons.OK);
        return null;
    }
    catch (System.UnauthorizedAccessException)
    {
        MessageBox.Show("Access to " + comPort + " denied. Please check if another program is accessing this port", "Error!", MessageBoxButtons.OK);
        return null;
    }
}
```

Step 2: Define Tag Read Event Listener to store tags in the software buffer –

```
StringDictionary associationDataBase = new StringDictionary();
TagReadData currentRead = null;
void PrintTagReadUsb(Object sender, TagReadDataEventArgs e)
{
     TagReadData read = e.TagReadData;
     if (!associationDataBase.ContainsKey(read.EpcString))
     {
          currentRead = read;
     }

}
```

Step 3: Read tags –

```
private void Read()
{
    rdr1.ReadException += delegate(Object senderException, ReaderExceptionEventArgs re)
    {
        MessageBox.Show("Error: " + re.ReaderException.Message.ToString());
    };
    rdr1.TagRead += PrintTagReadUsb;
    rdr1.StartReading();
}
```

Step 4: Associate tag EPCs to racers –

```
private void associate(string currentTag, string nameOfParticipant)
{
     if ((nameOfParticipant != "") && (currentTag != ""))
     {
          if (associationDataBase.ContainsKey(currentTag))
          {
               associationDataBase.Remove(currentTag);
               associationDataBase.Add(currentTag, nameOfParticipant);
          }
          else
          {
               associationDataBase.Add(currentTag, nameOfParticipant);
          }
          writeToFile(makeDictionaryIntoXML(associationDataBase), "output.xml", false);
     }
}

private string makeDictionaryIntoXML(System.Collections.Specialized.StringDictionary poDict)
{
    string sReturnXML = "";
    sReturnXML = "";
    foreach (System.Collections.DictionaryEntry oDictEntry in poDict)
    {
        sReturnXML = sReturnXML.ToString() + "\r\n<item key='" + CheckXMLValue(oDictEntry.Key.ToString()) + "'><![CDATA["
            + oDictEntry.Value.ToString() + "]]></item>";
    }

    if (sReturnXML.Trim().ToString() != "")
    {
        sReturnXML = "<?xml version='1.0'?>\r\n<root>" + sReturnXML.ToString() + "\r\n</root>";
    }

    return sReturnXML;
}
```

Mercury 6 Readers –

A. The codelet below shows how to configure the Mercury6 reader to perform continuous read operation using read plans.

Please note: this codelet is a guideline and will need to be modified to work in your environment.

Step 1: Configure the reader – The example below shows the configuration for a gen2 protocol tag. If the system includes IPX tags instead, the variable tagProto below should be – TagProtocol.IPX256 or TagProtocol.IPX64 depending on the type of IPX tags used.

```
private Reader initilizeReaders(string hostName)
{
    Reader rdr1 = null;
    try
    {
        //Define the antenna port list to use for the reads
        int[] antennas = new int[] { 1, 2, 3, 4 };
        TagProtocol tagProto = TagProtocol.GEN2;
        //Define a read plan that will be executed when the reader is reading
        SimpleReadPlan readPlan = new SimpleReadPlan(antennas, tagProto);
        //Create the reader object
        rdr1 = Reader.Create("tmr://" + hostName);
        rdr1.Connect();
        //Set a timeout the host waits for transport messages to and from the reader
        //Useful for slow processors
        rdr1.ParamSet("/reader/transportTimeout", 5000);
        //Set asynchOffTime to 0 to enable continuous streaming mode of operation
        rdr1.ParamSet("/reader/read/asyncOffTime", 0);
        //Set the appropriate regulatory region setting for the region of operation
        Reader.Region regionToSet = Reader.Region.NA;
        rdr1.ParamSet("/reader/region/id", regionToSet);
        RqlReader rqlR = (RqlReader)rdr1;
        //Set the gen2 Session to 1. Allows reading of more than a single tag in the field
        //and does not put the tag to sleep for a long time so that the next read point can read the tag.
        rqlR.ParamSet("/reader/gen2/Session", Gen2.Session.S1);
        //Set the gen2 tag encoding scheme to the fastest data rate
        rqlR.ParamSet("/reader/gen2/tagencoding", Gen2.TagEncoding.FM0);
        //Set the gen2 backscatter link frequency to the fastest link rate
        rqlR.ParamSet("/reader/gen2/blf", Gen2.LinkFrequency.LINK640KHZ);
        rqlR.ParamSet("/reader/read/plan", readPlan);
        //Set the read power upto 3150cdBm depending on choice of antenna and area RF regulations
        rqlR.ParamSet("/reader/radio/readPower", 3000);
        return rdr1;
    }
    catch (System.IO.IOException)
    {
        MessageBox.Show("Reader not connected on " + hostName, "Error!", MessageBoxButtons.OK);
        return null;
    }
```

```
    catch (ReaderCodeException ex)
    {
        MessageBox.Show("Error connecting to Reader: " + ex.Message.ToString(), "Error!", MessageBoxButtons.OK);
        return null;
    }
    catch (System.UnauthorizedAccessException)
    {
        MessageBox.Show("Access to " + hostName + " denied. Please check if another program is accessing this port",
            "Error!", MessageBoxButtons.OK);
        return null;
    }
    catch (ReaderCommException)
    {
        MessageBox.Show("Reader not connected on " + hostName, "Error!", MessageBoxButtons.OK);
        return null;
    }
}
```

Step 2: Define Tag Read Event Listener to store tags in the software buffer –

```
Dictionary<string, TagReadData> tagDataBaseForM6 = new Dictionary<string, TagReadData>();
void PrintTagReadM6(Object sender, TagReadDataEventArgs e)
{
    TagReadData read = e.TagReadData;
    TagReadData forDataBase;
    string singulationString = "";
    singulationString = read.EpcString;
    lock (tagDataBaseForM6)
    {
        if (tagDataBaseForM6.Count == 0)
        {
            tagDataBaseForM6.Add(singulationString, read);
        }
        else
        {
            if (tagDataBaseForM6.ContainsKey(singulationString))
            {
                forDataBase = tagDataBaseForM6[singulationString];
                tagDataBaseForM6.Remove(singulationString);
                read.ReadCount += forDataBase.ReadCount;
                tagDataBaseForM6.Add(singulationString, read);
            }
            else
            {
                tagDataBaseForM6.Add(singulationString, read);
            }
        }
    }
}
```

Step 3: User has to define a function to connect to the central database and feed it with tagDataBaseForM6 populated in the above codelet.

Step 4: Read tags using StartReading() that initiates asynchronous continuous reading mode.

```
private void Read()
{
    rqlR.ReadException += delegate(Object senderException, ReaderExceptionEventArgs re)
    {
        MessageBox.Show("Error: " + re.ReaderException.Message.ToString());
    };
    rqlR.TagRead += PrintTagReadM6;
    rqlR.StartReading();
}
```

B. The codelet below shows how to perform writing data to a tag while reading other RFID tags.

Please note: this codelet is a guideline and will need to be modified to work in your environment.

This is achieved by modifying the initialize process to include the write data operation. The continuous streaming mode is not supported with embedded tag operations and the reader enters a pseudo asynchronous mode where it performs reads and writes in busts with negligible RF off time between operations.

```
private Reader initilizeReaders(string hostName)
{
    Reader rdr1 = null;
    try
    {
        //Define the antenna port list to use for the reads
        int[] antennas = new int[] { 1, 2, 3, 4 };
        //Define the operation to perform while reading tags
        Gen2.WriteData writeCheckpointCoordinates = new Gen2.WriteData(Gen2.Bank.USER, 0, ConvertCheckpointToByteArray());
        TagProtocol tagProto = TagProtocol.GEN2;
        //Define a read plan that will be executed when the reader is reading
        //and perform the write operations on the tags as they are read
        //A filter could also be added to perform write on a specific tag
        SimpleReadPlan readPlan = new SimpleReadPlan(antennas, tagProto, null, writeCheckpointCoordinates, 1000);
        //Create the reader object
        rdr1 = Reader.Create("tmr://" + hostName);
        rdr1.Connect();
        //Set a timeout the host waits for transport messages to and from the reader
        //Useful for slow processors
        rdr1.ParamSet("/reader/transportTimeout", 5000);
        //Set asynchOffTime to a non-zero timeout (ms) to enable pseudo asynch mode of operation
        rdr1.ParamSet("/reader/read/asyncOffTime", 10);
        //Set the appropriate regulatory region setting for the region of operation
        Reader.Region regionToSet = Reader.Region.NA;
        rdr1.ParamSet("/reader/region/id", regionToSet);
        RqlReader rqlR = (RqlReader)rdr1;
        //Set the gen2 Session to 1. Allows reading of more than a single tag in the field
        //and does not put the tag to sleep for a long time so that the next read point can read the tag.
        rqlR.ParamSet("/reader/gen2/Session", Gen2.Session.S1);
        //Set the gen2 tag encoding scheme to the fastest data rate
        rqlR.ParamSet("/reader/gen2/tagencoding", Gen2.TagEncoding.FM0);
        //Set the gen2 backscatter link frequency to the fastest link rate
        rqlR.ParamSet("/reader/gen2/blf", Gen2.LinkFrequency.LINK640KHZ);
        rqlR.ParamSet("/reader/read/plan", readPlan);
        //Set the read power upto 3150cdBm depending on choice of antenna and area RF regulations
        rqlR.ParamSet("/reader/radio/readPower", 3000);
        return rdr1;
    }
```

```
    catch (System.IO.IOException)
    {
        MessageBox.Show("Reader not connected on " + hostName, "Error!", MessageBoxButtons.OK);
        return null;
    }
    catch (ReaderCodeException ex)
    {
        MessageBox.Show("Error connecting to Reader: " + ex.Message.ToString(), "Error!", MessageBoxButtons.OK);
        return null;
    }
    catch (System.UnauthorizedAccessException)
    {
        MessageBox.Show("Access to " + hostName + " denied. Please check if another program is accessing this port",
            "Error!", MessageBoxButtons.OK);
        return null;
    }
    catch (ReaderCommException)
    {
        MessageBox.Show("Reader not connected on " + hostName, "Error!", MessageBoxButtons.OK);
        return null;
    }
}
```

```
private ushort[] ConvertCheckpointToByteArray()
{
    ushort[] returnData = new ushort[] { 0xFF, 0xFF };
    //Code to retrieve Checkpoint co-ordinates and converting to
    //ushort array
    return returnData;
}
```

Step2: Perform reading of tags as shown in Section A - Steps 2 to Step 4 once the write data operation is defined as shown above.

Summary

Races of all types can be timed by hand with operators using a stop-watch, or by using a combination of electronic timing and video camera systems. As with many time-sensitive activities however, UHF RFID has proven to be a more efficient alternative to manual tracking by reducing human error and providing the ability to process a greater amount of data in a shorter period of time.

As noted above, with multiple of areas of race timing benefiting from RFID, reader form factor is an important consideration. With ThingMagic broad portfolio of fixed/finished and embedded UHF RFID readers, one can develop highly integrated end-to-end race timing solutions. Equally important is tag read performance, including the RFID reader's ability to continuously read moving tags in a densely populated field, a high tag read rate, and a configurable read range. The ability to easily integrate RFID data into back-end applications should also be considered when selecting RFID readers for your race timing solution.

In addition to the applications noted above, UHF RFID can also be used to enhance the racer and spectator experience during the event. Through the combination of RFID-enabled tags and check points, sponsors can display messages on big screens for a given group of racers (e.g. employees of a particular company). Further, personal motivational messages could be displayed on screens for friends and family

participating in the race or real-time statistical information of a particular racer could be presented at different check points, giving racers an idea of their performance in relation to the rest of the field. Many of these innovative features can only be achieved in real-time through the use of UHF RFID.

RFID-enabled race timing solutions are offered by companies around the globe, including several ThingMagic partners. For more information, visit www.thingmagic.com.