NMV2D

Passive UHF RFID 915MHz, EPC Gen2v2 and ISO/IEC 29167-10 with I²C Interface and Temperature Sensor

APLICATION NOTE

HOW TO GET TEMPERATURE FROM NMV2D CHIP USING RFID READER

Overview

NMV2D is a semiconductor device compatible with EPCglobal ® Gen2v2 – the most recent industry international standard for RFID applications with security features, oriented to goods, identification, tracking and authentication.

This semiconductor device also has an integrated temperature sensor. Its usage is simple – execute a device setup for an accurate measurement and perform some Gen2 read operations at specific memory locations to get the raw data and convert them to a ºC temperature information.

This document will guide you to the required steps for this temperature processing using our NMV2D.

1. INTRODUCTION

Temperature measurement using NMV2D chip comprises 3 steps:

1. Inventory round followed by a **Read operation of the calibrated parameters** (values named as **A**, **B** and **C**) stored at three memory word addresses of the chip, each one to be used in the temperature equation calculation;
2. Write a dummy data into a non-usable memory address to **configure the chip internal power supply** for an accurate temperature measurement procedure;
3. **Read the temperature code** (**N**) using a Gen2 Read command to a specific memory address;
4. **Get the measured current temperature** through an **equation calculation** using the gotten values from previous steps.

> **IMPORTANT**:
>
> During steps 2 and 3, **RF signal must be kept turned-on** (**CW - Continuous Wave** option) between the RF commands.

Follow next instructions to understand the steps that must be added to your RFID Reader's software or its controller before executing current temperature measurement.

2. GETTING CALIBRATED PARAMETERS (VALUES A, B AND C)

Three data used as equation parameters required for temperature calculation are stored at USER's memory bank (3_h).

Descriptions for the 3 data are:

- **A** – Curvature parameter for temperature calculation (word address $1D_h$ of USER's bank)
- **B** – Resolution parameter for temperature calculation (word address $1E_h$ of USER's bank)
- **C** – Minimal temperature measurable parameter (word address $1F_h$ of USER's bank)

So, **perform the standard UHF Read** operation following the specification below to read all the 3 calibrated parameters:

Table 1 – Special memory address reading setup for current temperature measurement

	Command**	*MemBank**	*WordPtr**	*WordCount*	*RN***	*CRC***
# of bits	8	2	8	8	16	16
definition (binary)	11000010_2	11_2: USER	00011101_2	00000011_2	*handle*	*CRC-16*
(hexadecimal)	$C2_h$	3_h	$1D_h$	03_h		

* Following Gen2 RFID Standard
** See Gen2 RFID Standard for more details about these parameters. RFID Readers implements these parameters according to the standard and users do not have to care about them.

The 48-bit data acquired (3 words) from the standard UHF Read operation defined above is compounded as is:

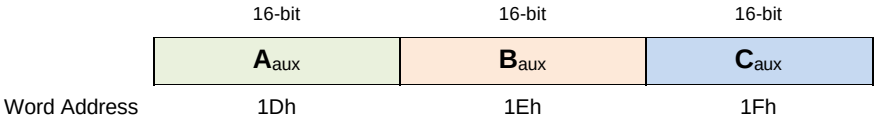

Figure 1 – NMV2D parameters for temperature measurement

These read data must be adjusted (multiplied to exponentials) to be used as the A, B and C parameters in the final temperature equation.

- **A** = A_{aux} *** 10^{-8}**
- **B** = B_{aux} *** 10^{-5}**
- **C** = C_{aux} *** 10^{-2}**

For example, if the data read from NMV2D is 0x114C4FD4165B (in hexadecimal), parameters A, B and C are:

- A_{aux} = 0x114C
- B_{aux} = 0x4FD4
- C_{aux} = 0x165B

And then:

- **A** = A_{aux} *** 10^{-8}** = 0x114C (4428 in decimal) * 10^{-8} = 4428 * 10^{-8} = **$44.28e^{-6}$**
- **B** = B_{aux} *** 10^{-5}** = 0x4FD4 (20436 in decimal) * 10^{-5} = 20436 * 10^{-5} = **$204.36e^{-3}$**
- **C** = Caux *** 10^{-2}** = 0x165B (5723 in decimal) * 10-2 = 5723 * 10^{-2} = **57.23**

3. CONFIGURING THE CHIP INTERNAL POWER SUPPLY

First, after a standard UHF inventory phase (Select command) it is required to prepare the NMV2D chip internal power supply circuit **executing 8 successful UHF Write operations** on a safety memory region as specified by the next table:

Table 2 – Special memory address writing setup for NMV2D internal power supply alignment

	Command**	*MemBank**	*WordPtr**	*Data*	*RN***	*CRC***
# of bits	8	2	8	16	16	16
definition (binary)	11000011_2	11_2: USER	00000000_2	00000000_2	*handle*	*CRC-16*
(hexadecimal)	$C3_h$	3_h	00_h	0000_h		

* Following Gen2 RFID Standard
** See Gen2 Standard for more details about these parameters. RFID Readers implements these parameters according to the standard and users do not have to care about them.

To certify that the NMV2D chip internal power supply is correctly configured, check the Write command reply parameter named *Header* at each execution, that must be equal to 0_h, indicating a successful operation done.

Table 3 - NMV2D reply to a successful Write command

	Header	*RN**	*CRC**
# of bits	1	16	16
description (binary)	0_2	*handle*	*CRC-16*
(hexadecimal)	0_h		

** See Gen2 Standard for more details about these parameters. RFID Readers implements these parameters according to the standard and users do not have to care about them.

If you receive 8 successful write operations in a sequence, NMV2D chip is ready for the next steps. If not, adjust the RFID Reader power and other available configurations to reach the required conditions for an accurate temperature measurement.

Again, the RFID Reader must not power down at any time between this step and the next (standard UFH commands defined here).

4. GETTING TEMPERATURE CODE (N)

To get the temperature code from the NMV2D integrated sensor, you must execute a standard UHF **Read command** to a specific memory address.

Table 4 – Special memory address reading setup for current temperature measurement

	Command**	*MemBank**	*WordPtr**	*WordCount*	*RN***	*CRC***
# of bits	8	2	8	8	16	16
definition (binary)	11000010_2	00_2: RESERVED	00001000_2	00000001_2	*handle*	*CRC-16*
(hexadecimal)	$C2_h$	0_h	08_h	01_h		

* Following Gen2 RFID Standard
** See Gen2 RFID Standard for more details about these parameters. RFID Readers implements these parameters according to the standard and users do not have to care about them, just provide the values for the parameters at command line/software

The 16-bit data acquired from the standard UHF Read command defined above is compounded as is:

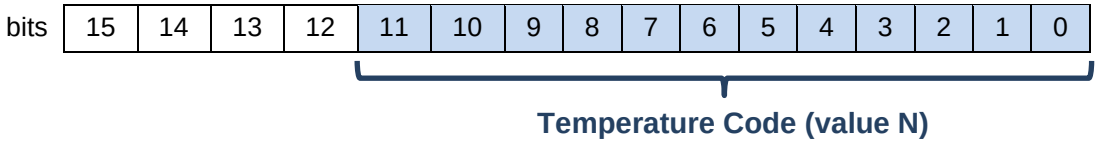

Figure 2 - NMV2D current temperature measurement data composition

For example, if the data read from NMV2D is 0x31C5 (in hexadecimal), **N** value for this read operation is **0x1C5** (**453** in decimal format).

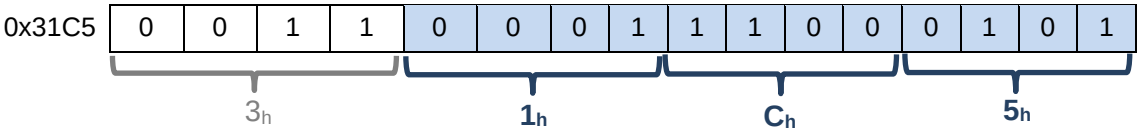

Figure 3 - Example of temperature code read from NMV2D

Execute this standard UHF Read command **10 times** for an accurate measurement. Exclude the minimum and maximum values and consider **N** (temperature code) as the average of the remaining data.

IMPORTANT:

REJECT readings with N = 0x000 (000_h)

5. CALCULATING THE CURRENT TEMPERATURE

Finally, to get the current temperature measurement, use the 2nd order equation:

$$\mathbf{T = -A * N^2 + B * N - C}$$

Considering the previous example values in decimal format (N = 453; A = $44.28e^{-6}$; B = $204.36e^{-3}$; C = 57.23), we have:

$$\mathbf{T} = -A*N^2 + B*N - C$$

$$\mathbf{T} = -44.28e^{-6} * (453)^2 + 204.36e^{-3} * 453 - 57.23$$

$$\mathbf{T} = 26.26\ ^oC$$

6. SAMPLE CODE

The sample Python code below illustrates the libraries and other definitions to be used as bases for the other Python codes.

```
# ============== Libraries ======================================================

from pyftdi import FtdiLogger
from pyftdi.ftdi import Ftdi
from pyftdi.i2c import I2cController, I2cNackError
from pyftdi.misc import add_custom_devices
import time
import datetime
import binascii
import telnetlib
import re
import sys
import os
from statistics import mode
from statistics import mean


# ======== INTERMEC IF2 RFID Reader Setup ======================================

rfid_HOST = '192.168.4.11'
rfid_PORT = 2189
IF2Error = ['NOTAG', 'WRERR']
rfid_PROMPT = 'OK>'


def tn_connect(tn_host: str, tn_port: int, timeout: int=1) -> telnetlib.Telnet:
    """ Connect via Telnet to IP HOST
        :param tn_host: Host IP address, string
        :param tn_port: Host Port, int
        :param timeout: Timeout to stablish connection
        :return tn: a Telnet instance,
                    return 'None' if no connection stablished
    """
    try:
        tn = telnetlib.Telnet(tn_host, tn_port, timeout)
        ans = tn.read_until(b'>',1)
        ans = tn.read_eager()
    except:
        tn = None
        print('ERROR TO CONNECT TO HOST:',tn_host,tn_port)
    return tn
```

```python
def tn_send(tn, cmd: str, prt_cmd: bool=False, timeout: int=10) -> str:
    """ Send Command to TELNET
        :param tn : telnetlib.Telnet instance
        :param cmd: Command to be statements, string
        :param prt_cmd: True for print cmd and reply, False not print
        :return Ans: Answers from TELNET, string

        Note:
        >>> tn_send(rfid, 'r \n')
        H0000000000000201810120108H003C0A00002A
        (encode ascii) b'\r\nH0000000000000201810120108\r\nH003C0A00002A'
    """
    if tn is None:
        print('PORT not defined')
        return ''
    elif tn == rfid:
        PROMPT = rfid_PROMPT
        CMD_SRC = 'RFID>'
    tn.write(cmd.encode('ascii'))
    ans = tn.read_until(PROMPT.encode('ascii'),timeout)
    ans = ans.decode('ascii').replace(PROMPT,'') # Remove PROMPT
    ans = re.sub('(^\r\n)|(\r\n$)','',ans) # Remove '\r\n' at start and end
    if prt_cmd :
        print(CMD_SRC, cmd.replace('\n',''), end = '')
        print('\r\n>', ans) if len(ans) else print('')
    return ans


def rfid_Inv() -> str:
    """ RFID INVENTORY to select and return a TAG EPC target
        :return EPC: TAG EPC selected, str
    """
    print('\n>>> Run INVENTORY <<<\n')

    for i in range(5):
        tn_send(rfid, 'r\n')
        ans = tn_send(rfid, 'repeat 10\n')
        if (ans.count('NOTAG') == 0) :
            break
    inv = dict()
    for word in ans.split():
        inv[word] = inv.get(word,0) + 1
    i = 0
    inv_list = dict()
    if len(inv) == 0:
        return None
    print('index : EPC | Count')
    for word, count in inv.items():
        inv_list[i] = word
        print(i,'    :',word,'|',count)
        i += 1
    if len(inv_list) == 1: return inv_list[0]
    while(1):
        try:
            idx = int(input('\n Select EPC by index: '))
            print('')
            return inv_list[idx]
        except KeyboardInterrupt:
            print('[ESC] END by User ...')
            exit()
        except:
            print('ERR: Please enter a valid index [0 : %d]'%(len(inv)-1))
            continue
```

The sample Python code below illustrates how to define functions to configure the NMV2D chip internal power supply using **Intermec IF2 RFID Reader**.

```
def rfid_WriteData(epc: str='') -> int:
    """ DUMMY WRITES to check enough RF Power received by DUT
        :param epc: EPC from target TAG, ex: 'H00CAFE', string
        :return   : Count of WROK, expected 8, int
    """
    PRT = False
    sel ='where hex(1:4,%d)=%s'%((len(epc)-1)/2,epc) \
        if epc.startswith('H') else ''
    cmd = ('write hex(3:0,2)=H0000 hex(3:0,2)=H0000 hex(3:0,2)=H0000 \
hex(3:0,2)=H0000 hex(3:0,2)=H0000 hex(3:0,2)=H0000 hex(3:0,2)=H0000 \
hex(3:0,2)=H0000 %s\n'%(sel))
    ans = tn_send(rfid, cmd, PRT)
    ans = ans.split()
    return ans.count('WROK')


def rfid_TunePwr(epc: str='') -> int:
    """ TUNE POWER PROCESS to guarantee enough RF Power to get Count from TAG
        :param epc: EPC from target TAG, ex: 'H00CAFE', string
        :return   : Power value configurated in dBm, int
    """
    print('\n>>> Tunning RF Power <<<')
    PwMin = 1
    PwMax = 30
    PwStep = 3
    for pw in range(PwMin, PwMax, PwStep):
        tn_send(rfid, 'ATTRIB FIELDSTRENGTH=1dB,1dB,1dB,%sdB\n'%(pw))
        time.sleep(1)
        wok = rfid_WriteData(epc)
        print('[TEST] PWR[dBm]: %d, #WROK: %d/8'%(pw, wok))
        if (wok == 8) : break
    if pw < 30: pw += 1
    tn_send(rfid, 'ATTRIB FIELDSTRENGTH=1dB,1dB,1dB,%sdB\n'%(pw), True)
    return pw
```

The sample Python code below illustrates how to define functions to read the calibrated parameters (values A, B and C) from NMV2D chip using **Intermec IF2 RFID Reader**.

```
def rfid_ReadABC(epc: str='') -> list:
    """ Read TEMP COUNTER from target TAG
        :param epc: EPC from target TAG, ex: 'H00CAFE', str
        :return array: TEMP 3 PARAMETERS ABC, ex:[], list float
    """
    PRT = False
    sel ='where hex(1:4,%d)=%s'%((len(epc)-1)/2,epc) \
        if epc.startswith('H') else ''
    cmd = ('r hex(3:58,6) %s\n'%(sel))
    ABC = []
    for i in range(5):
        ans = tn_send(rfid, cmd, PRT)
        ans = ans.replace(epc, '')
        ans = re.sub(' H','',ans)
        if len(ans) == 12:
            ABC = [int(ans[i:4+i],16) for i in range(0, len(ans), 4)]
            print('\n[INFO] READ hex(3:58,6): H%s [DEC]:'%(ans),ABC)
            ABC = [ABC[0]*1e-8,ABC[1]*1e-5,ABC[2]*1e-2]
            break
    if len(ABC) == 0:
        print('[ERR] NO PARAM READED!!!')
        print('exit by program')
        exit()
    return ABC
```

The sample Python code below illustrates how to read the temperature code (value N) from NMV2D chip using **Intermec IF2 RFID Reader**.

```
def rfid_ReadCount(epc: str='') -> str:
    """ Read TEMP COUNTER from target TAG
        :param epc: EPC from target TAG, ex: 'H00CAFE', str
        :return data: TEMP COUNTER N, ex:'H215A', str
    """
    PRT = False
    sel ='where hex(1:4,%d)=%s'%((len(epc)-1)/2,epc) \
        if epc.startswith('H') else ''
    cmd = ('r hex(0:16,2) %s\n'%(sel))
    ans = tn_send(rfid, cmd, PRT)
    ans = tn_send(rfid, 'repeat 10\n')
    ans = ans.replace(epc, '')
    ans = re.sub('H.','',ans)
    Nlist = []
    for line in ans.split():
        Nlist.append(int(line,16))
    return round(mean(Nlist))
```

Finally, the main Python code to calculate the temperature in ºC using new and previous defined functions and general definitions, the integrated temperature sensor at NMV2D chip and an **Intermec IF2 RFID Reader**.

```
def rfid_Config() -> None:
    """ Configure RFID Reader IF2
    """
    PRT = True
    print ('\n>>> Configure RFID Reader IF2... <<<\n')
    ans = tn_send(rfid, 'ver\n')
    print(ans,'\r\n')
    tn_send(rfid, 'ATTRIB echo=OFF\n', PRT)
    tn_send(rfid, 'ATTRIB epcc1g2parms = 12\n', PRT)
    tn_send(rfid, 'ATTRIB schedopt     = 2\n', PRT)
    tn_send(rfid, 'ATTRIB anttimeout   = 50\n', PRT)
    tn_send(rfid, 'ATTRIB idtries      = 1\n', PRT)
    tn_send(rfid, 'ATTRIB anttries     = 2\n', PRT)
    tn_send(rfid, 'ATTRIB UNSELTRIES   = 0\n', PRT)
    tn_send(rfid, 'ATTRIB rdtries      = 4\n', PRT)
    tn_send(rfid, 'ATTRIB wrtries      = 4\n', PRT)
    tn_send(rfid, 'ATTRIB idreport     = on\n', PRT)
    tn_send(rfid, 'ATTRIB idtimeout    = 100\n', PRT)
    tn_send(rfid, 'ATTRIB session      = 0\n', PRT)
    tn_send(rfid, 'ATTRIB INITIALQ     = 0\n', PRT)
    tn_send(rfid, 'ATTRIB ants=4\n', PRT)
    tn_send(rfid, 'ATTRIB fs=1dB,1dB,1dB,15dB\n', PRT)



#####################################################################
#############      MAIN          ###################################
#####################################################################

start_time = time.time()
rfid = tn_connect(rfid_HOST, rfid_PORT)
rfid_Config()
epc = rfid_Inv()

pw1 = rfid_TunePwr(epc)
ABC = rfid_ReadABC(epc)

summary = 'DUT EPC: %s\n\n'%epc

print('\n-----------------------------------------------------------')
print('PARAMETERS ABC\nT(°C) = -A*N^2 + B*N - C')
print('\n[     A     |     B     |    C    ]')
print('[ %.3E | %.5f | %.2f ]'%(ABC[0], ABC[1], ABC[2]))
print('-----------------------------------------------------------\n')
st_time = time.time()
```

```
input('\nPRESS << ENTER >> TO RUN TEMPERATURE MONITOR ')
print('\n-----------------------------------------------------')
print('Timestamp > [Temperature TAG] [N TAG] ')
print('-----------------------------------------------------')
while(1):
    Ndut = rfid_ReadCount(epc)
    Tdut = (-ABC[0]*Ndut*Ndut + ABC[1]*Ndut - ABC[2])
    elap_time = time.strftime('%H:%M:%S', time.gmtime(time.time()-st_time))
    print('%s> T_TAG= %.4f [N= %d]'\
        %(elap_time, Tdut, Ndut))
```

CONTACT INFORMATION

www.wvblabs.com
contact@wvblabs.com

Campinas, São Paulo, Brazil
(Headquarters/Labs)
Av. Alice de C. Pupo Nogueira Mattosinho, 301
Alphaville, 13098-392
Phone: +55 (19) 3756-5454

São Carlos, São Paulo, Brazil
(Labs)
R. Dom Pedro II, 2085 – offices 21, 22, 23, 24 and 25
Jd. Macarenco, 13560-320
Phone: +55 (16) 3307-8305

NOTICES