Transparent Mode on PT1xxR Family

The Transparent Mode API for the PT1xxR family of NFC devices enables an application to implement proprietary protocols based on low-level RF commands.

Contents

Figures

1. Requirements

This document applies to:

- POS/IoT SDK for PTX1xxR family v7.2.0

2. Transparent Mode API Functions

The following subsections present source code snippets in C to show how to use the Transparent Mode's API functions. For more information about the general usage of the POS SDK, see the integration note located in the "*DOCS*" folder of the SDK release package.

2.1 Prerequisites

Transparent Mode can only be activated in the READY state of the system. It switches to TEST state after calling ptxTransparentMode_SetField as shown in Figure 1. Additional information is located in the SDK integration notes of POS/IoT SDKs. For more information on the ptxTransparentMode_SetField, see section 2.3.

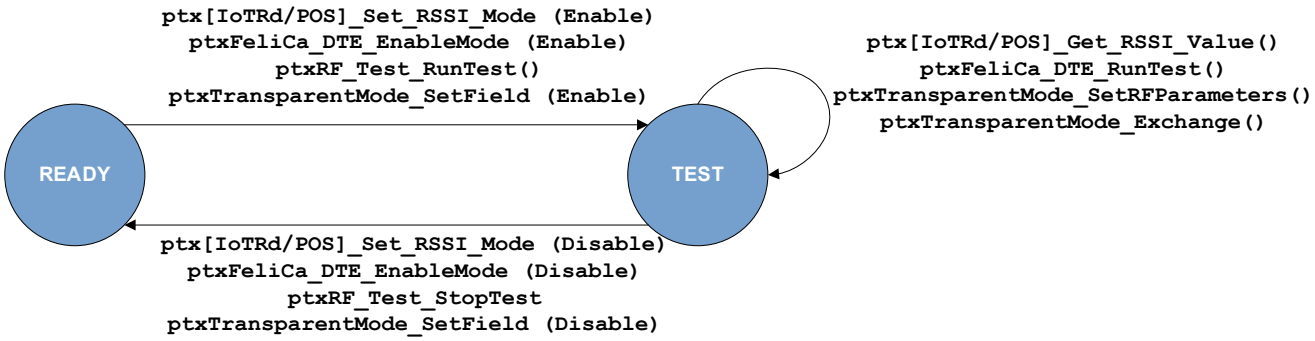

Figure 1. State Transitions form READY to TEST

2.2 ptxTransparentMode_Init

ptxTransparentMode_Init initializes the basic prerequisites in order to let the PTX1xxR operate in the transparent mode.

2.2.1 API Signature

In Source Code 1, you can see the signature of the function. The user must provide pointers to the applicable structures for the component itself and the initialization parameters. The ptxTransparentMode_t structure contains pointers to a logger, OSAL struct, completion status, and others.

```
1  /**
2   * \brief Initializes the Transparent-Mode Component.
3   *
4   * \param[in]   tmComp      Pointer to an existing instance of the
    Transparent-Mode component.
5   * \param[in]   initParams  Pointer to initialization parameters.
6   *
7   * \return Status, indicating whether the operation was successful.
8   */
9  ptxStatus_t ptxTransparentMode_Init (ptxTransparentMode_t *tmComp,
   ptxTransparentMode_InitParams_t *initParams);
```

Source Code 1. ptxTransparentMode_Init Signature

2.2.2 API Usage

Source Code 2 shows an example of how to get the PTX1xxR ready to initialize the transparent mode.

```
ptxStatus_t st = PTX_STATUS_UNDEFINED;

/*
 * Please refer the POS SDK integration note until the ptxPOS_Init_NSC method
 * and then continue with the code example provided
 */

//Initializes the PTX1xxR.
nsc_init_config.CalibratedTempThreshold = app_thermal_shutdown;

/* Read RF-Configuration from .dat-File (Note: RfConfigParameter is optional; NULL means to read from file) */
nsc_init_config.RfConfigParameter = NULL;

st = ptxIoTRd_Init_NSC(stack_comp, &nsc_init_config);
if (PTX_STATUS_OK == st)
{

    ptxCommon_PrintF ("Successful initialization of the NSC. \n");

    /*
        * Print all revisions provided by the system.
        */
    ptxIoTRdInt_Print_Revision_Info(stack_comp);


    if (PTX_STATUS_OK == st)
    {
        ptxCommon_PrintF("OK\n");

        ptxTransparentMode_t tm_comp = {0};

        ptxTransparentMode_InitParams_t init_params = {0};
        init_params.StackComp = stack_comp;

        st = ptxTransparentMode_Init(&tm_comp, &init_params);
```

Source Code 2. ptxTransparentMode_Init Usage

2.3 ptxTransparentMode_SetField

This enables or disables the Transparent Mode by turning the RF field on or off.

2.3.1 API Signature

Source Code 3 shows the signature of the function. The user must provide a pointer to the same transparent mode component that was used for ptxTransparentMode_Init and an integer to turn it on or off.

```
/**
 * \brief Turns the RF-field on or off.
 *
 * \param[in]       tmComp  Pointer to an existing instance of the
Transparent-Mode component.
 * \param[in]       state   State of RF-field (0 = off, != 0 = on).
 *
 * \return Status, indicating whether the operation was successful.
 */
ptxStatus_t ptxTransparentMode_SetField (ptxTransparentMode_t *tmComp,
uint8_t state);
```

Source Code 3. ptxTransparentMode_SetField Signature

2.3.2 API Usage

An application must call this function before any further call to ptxTransparentMode_SetRFParameters and/or ptxTransparentMode_Exchange with input parameter state ≥1 (RF on). To turn the field off again, use state = 0.

```
//turn field on
st = ptxTransparentMode_SetField(&tm_comp, 1);
//turning it off
st = ptxTransparentMode_SetField(&tm_comp, 0);
```

Source Code 4. ptxTransparentMode_SetField Usage

2.4 ptxTransparentMode_SetRFParameters

This API function can be used to configure the HW for the following call(s) to ptxTransparentMode_Exchange. The configuration parameters in Table 1 are supported.

Table 1. ptxTransparentMode_SetRFParameters' Configuration Parameters

RF/Card-Technology	A, B, F or V
Tx- and Rx-Bitrates	106 / 212 / 424 / 848 / 26.5 kBit/s
Flags (Parity, CRC)	Enable / Disable Tx-/Rx-CRC handling Enable / Disable Tx-/Rx-Parity-Bit handling
Number of Tx-Bits	Number of (residual) Bits to be sent for last byte
RES-Limit	Maximum number of responses to receive

2.4.1 API Signature

Source Code 5 shows the signature of the function. The user must provide a pointer to the same transparent mode component that was used for ptxTransparentMode_Init and a pointer to a RF-parameters struct.

```
/**
 * \brief Configures the HW using the provided RF-Parameters (used for all following RF-Exchanges, can be overwritten).
 *
 * \param[in]   tmComp                  Pointer to an initialized instance of the Transparent-Mode component.
 * \param[in]   rfParams                Pointer to RF-parameters.
 *
 * \return Status, indicating whether the operation was successful.
 */
ptxStatus_t ptxTransparentMode_SetRFParameters (ptxTransparentMode_t *tmComp, ptxTransparentMode_RFParams_t *rfParams);
```

Source Code 5. ptxTransparentMode_SetRFParameters signature

2.4.2 API Usage

Source Code 1 shows an example of how the user can initialize the RF-parameters struct and use it as an argument for ptxTransparentMode_SetRFParameters.

```
ptxTransparentMode_RFParams_t transparent_mode_rf_params;
transparent_mode_rf_params.Tech = TM_RF_Tech_A;
transparent_mode_rf_params.TxRate = TM_RF_Bitrate_106;
transparent_mode_rf_params.RxRate = TM_RF_Bitrate_106;
transparent_mode_rf_params.NrTxBits = 7;
transparent_mode_rf_params.ResLimit = 1;
transparent_mode_rf_params.Flags = (uint8_t)(PTX_TRANSPARENT_MODE_FLAGS_TX_PARITY | PTX_TRANSPARENT_MODE_FLAGS_RX_PARITY);
st = ptxTransparentMode_SetRFParameters(&tm_comp, &rf_params);
```

Source Code 1. ptxTransparentMode_SetRFParameters Usage

2.5 ptxTransparentMode_Exchange

This API function performs the actual data exchange via RF between the card/emulated card and the PTX1xxR.

2.5.1 API Signature

Source Code 7 shows the signature of the function. The user must provide pointers to the same transparent mode component that was used for ptxTransparentMode_Init, RF parameters of ptxTransparentMode_SetRFParameters, and an integer to set the timeout duration (Note: As a safety feature, ~500ms are added in the function).

In addition, the following resources are required:

- A pointer to the data, which is intended to be transferred
- A pointer to a buffer that should store the received answer
- The length/size of both

```
/**
 * \brief Performs a RF data exchange.
 *
 * Attention: The last received byte is the contactless status byte which is defined as follows:
 *            Bit 7:     If set to 1, a contactless error occured (e.g. CRC- or Parity-error).
 *            Bit 6 - 0: Number of valid bits in last received byte.
 *
 * \param[in]       tmComp              Pointer to an initialized instance of the Transparent-Mode component.
 * \param[in]       rfParams            Pointer to RF-parameters (optional, otherwise parameters from \ref ptxTransparentMode SetRFParameters are used).
 * \param[in]       tx                  Buffer containing the data to send.
 * \param[in]       txLength            Length of "tx".
 * \param[out]      rx                  Pointer to buffer where the data will be received.
 * \param[in,out]   rxLength            As input, capacity of "rx". As output, actual number of bytes written on "rx".
 * \param[in]       timeoutMS           Timeout given in [ms].
 *
 * \return Status, indicating whether the operation was successful.
 */
st = ptxTransparentMode_Exchange(&tm_comp, &rf_params, &tx[0], tx_len, &rx[0], &rx_len, timeout);
```

Source Code 7. ptxTransparentMode_Exchange Signature

2.5.2 API Usage

In Source Code 8, an example shows how to initialize the buffers and use them for ptxTransparentMode_Exchange. Furthermore, received data always contains one additional byte which contains a contactless status byte.

The contactless status byte is defined as follows:

- Bit 7: If set to 1, a contactless error occurred (for example, CRC- or Parity-error).
- Bit 6–0: Number of valid bits in last received byte.

Example: If the answer to a REQ-A / SENS_REQ is 0x44 0x03, the API function returns rxLength = 3 (2 Byte received data + 1 byte contactless status byte), rx = 0x44 0x03 0x00 (last byte = contactless status byte).

```
(void)memset(&rx[0], 0, sizeof(rx));
rx_len = sizeof(rx);
tx_len = 1u;
ptxCommon_PrintF("Tx = %02X, Nr Residual Bits = %d\n", RID_CMD[i],
    rf_params.NrTxBits);
st = ptxTransparentMode_Exchange(&tm_comp, &rf_params, &RID_CMD[i],
    (uint8_t)1, &rx[0], &rx_len, timeout);
```

Source Code 8. ptxTransparentMode_Exchange Usage

3. T1T (Topaz) Transparent Mode Example

As the T1T standard can be tricky RF-wise, it is used as an example here. It is assumed it may be easier to adapt it to simpler standards by the user than the other way around.

In summary, the following are the outstanding characteristics of T1T:

- Every byte in the Tx-direction is sent as a separate Type-A Frame without parity bit.
- Despite being Type A, the CRC of the Type B standard is used.
- Timeouts must be handled differently.

3.1 Explanation of the Example

Starting with line 1 of Source Code 9, the author recommends to read “Prerequisites” to be aware of what is necessary to use this example. Also, in case of a POS project, the user should start with `ptxNSC_Init()`. For information, see the applicable integration note and example program provided by the SDK.

Furthermore, it must be noted that users are welcome to evaluate the return status `st` in this example more often and in greater detail. This was avoided to keep the example compact and less complicated.

Looking at line 8 to 23 of Source Code 9, parameters and component structures are defined and in line 15 the initialization function of the transparent mode is called (fore more information, see “ptxTransparentMode_Init”). The RF field is turned on in line 27, followed by a short delay, in order to let the hardware get to the correct state.

In line 30-35 and 50-55 of Source Code 9, the default configuration values are set for sending T1T frames.

Despite the T1Ts having their own command set, communication must start with a REQ-A or WUP-A according to ISO 14443-3A. This is why in line 43, by using `ptxTransparentMode_Exchange()`, a WUP-A is sent (for more information, see “ptxTransparentMode_Exchange”). In the case of a successful communication, the response is printed in line 47 and 48.

The program continues by reading the T1T’s ID. Therefore, the command must be defined in line 62 and the Type B CRC calculated in line 63. The function body can be found in Source Code 10.

Sending the single bytes requires different parameters, especially the timeout. Therefore, the for-loop combined with a switch case statement, starting at line 67 of Source Code 9 is used. The very first byte (= T1T Command-byte), handled by `case 0` in line 71, must be sent with 7 (residual) bits. In addition, the timeout must be set to 1ms. This will lead to a RF/Timeout error that is expected here because there will only be a card response after the last transmitted byte.

Starting in line 76/`case 8`, the last byte of T1T complete command is handled. In this case timeout must be increased because the card is expected to answer.

The default case in line 81 takes care of every other byte (ADD, data, etc.) which must be sent as full byte with 8 bits. In addition, the timeout must be set to 1ms although this will lead to a RF/Timeout error that is expected here because will only be a card response after the last transmitted byte.

Via ptxTransparentMode_Exchange in line 91, the line before printed/shown data is sent (for more information, see “ptxTransparentMode_Exchange”) followed by an output of the received content of the RX buffer if there is some.

At the end in line 101 the RF field is turned off.

```
1  st = ptxIoTRd Init NSC(stack comp, &nsc init config);
2  if (PTX_STATUS_OK == st)
3  {
4      ptxCommon_PrintF ("Successful initialization of the NSC. \n");
5
6      if (PTX_STATUS_OK == st)
```

```
7       {
8           ptxCommon_PrintF("OK\n");
9
10          ptxTransparentMode_t tm_comp = {0};
11
12          ptxTransparentMode_InitParams_t init_params = {0};
13          init_params.StackComp = stack_comp;
14
15          st = ptxTransparentMode_Init(&tm_comp, &init_params);
16
17          ptxTransparentMode_RFParams_t rf_params = {0};
18
19          uint8_t tx[256];
20          uint8_t tx_len = 0;
21          uint8_t rx[256];
22          uint32_t rx_len = 0;
23          uint32_t timeout;
24
25          while (!exit_loop)
26          {
27              st = ptxTransparentMode_SetField (&tm_comp, 1u);
28              Sleep(10);
29
30              rf_params.Tech = TM_RF_Tech_A;
31              rf_params.NrTxBits = 7u;
32              rf_params.ResLimit = 1u;
33              rf_params.RxRate = TM_RF_Bitrate_106;
34              rf_params.TxRate = TM_RF_Bitrate_106;
35              rf_params.Flags = PTX_TRANSPARENT_MODE_FLAGS_RX_PARITY;
36
37              ptxCommon_PrintF("Send WUP-A\n");
38              timeout = 100;
39              rx_len = sizeof(rx);
40              tx_len = 1u;
41              tx[0] = 0x52;
42              ptxCommon_PrintF("Tx = %02X\n", tx[0]);
43              st = ptxTransparentMode_Exchange(&tm_comp, &rf_params, &tx[0],
    tx len, &rx[0], &rx len, timeout);
44
45              if ((0 != rx_len) && (0 == st))
46              {
47                  ptxCommon_PrintF("Rx = ");
48                  ptxCommon_Print_Buffer(&rx[0], 0, rx_len, 1u, 0);
49              }
50
51              Sleep(5);
```

```
52
53              ptxCommon_PrintF("T1T Read ID (RID)\n");
54
55              rf_params.Tech = TM_RF_Tech_A;
56              rf_params.NrTxBits = 7u;
57              rf_params.ResLimit = 1u;
58              rf_params.RxRate = TM_RF_Bitrate_106;
59              rf_params.TxRate = TM_RF_Bitrate_106;
60              rf_params.Flags = PTX_TRANSPARENT_MODE_FLAGS_RX_PARITY;
61
62              uint8_t RID_CMD[9] = {0x78, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00};
63              ComputeCrc(CRC_B, &RID_CMD[0], (int)(sizeof(RID_CMD) - 2),
    &RID_CMD[7], &RID_CMD[8]);
64
65              printf("=====================================\n");
66
67              for (int i = 0; i < sizeof(RID_CMD); i++)
68              {
69                  switch (i)
70                  {
71                      case 0:
72                          rf_params.NrTxBits = 7;
73                          timeout = 1;
74                          break;
75
76                      case 8:
77                          timeout = 100;
78                          rf_params.NrTxBits = 0;
79                          break;
80
81                      default:
82                          rf_params.NrTxBits = 0;
83                          timeout = 1;
84                          break;
85                  }
86
87                  (void)memset(&rx[0], 0, sizeof(rx));
88                  rx_len = sizeof(rx);
89                  tx_len = 1u;
90                  ptxCommon_PrintF("Tx = %02X, Nr Residual Bits = %d\n",
    RID_CMD[i], rf_params.NrTxBits);
91                  st = ptxTransparentMode_Exchange(&tm_comp, &rf_params,
    &RID_CMD[i], (uint8_t)1, &rx[0], &rx_len, timeout);
92
93                  if ((0 != rx_len) && (0 == st))
94                  {
```

```
                    ptxCommon_PrintF("Rx = ");
                    ptxCommon_Print_Buffer(&rx[0], 0, rx_len, 1u, 0);
                }
            }

            Sleep(50);
            st = ptxTransparentMode_SetField (&tm_comp, 0);
            Sleep(50);

            printf("=====================================\n");
        }
    }
}
```

Source Code 9. T1T (Topaz) Transparent Mode Example

```
static void ComputeCrc(int CRCType, uint8_t *Data, int Length, uint8_t
*TransmitFirst, uint8 t *TransmitSecond)
{
    uint8_t chBlock;
    uint16_t wCrc;

    switch(CRCType)
    {
        case CRC_A:
            wCrc = 0x6363; /* ITU-V.41 */
            break;

        case CRC_B:
            wCrc = 0xFFFF; /* ISO/IEC 13239 (formerly ISO/IEC 3309) */
            break;

        default:
            return;
    }

    do
    {
        chBlock = *Data++;
        UpdateCrc(chBlock, &wCrc);

    } while (--Length);

    if (CRCType == CRC_B)
        wCrc = ~wCrc; /* ISO/IEC 13239 (formerly ISO/IEC 3309) */

    *TransmitFirst = (uint8_t) (wCrc & 0xFF);
```

```
31      *TransmitSecond = (uint8_t) ((wCrc >> 8) & 0xFF);
32
33      return;
34  }
```

Source Code 10. ComputeCrc

4. RF Signal Examples

The following sections show examples of commands for each of the four NFC modulation types. However, the scope of these examples is limited to the poller to listener modulation.

For information on the modulations, see the “NFC-A/B/F/V Technology” chapters in the *NFC Forum’s Digital Protocol Technical Specification* and the *NFC Forum’s Analog Technical Specification*.

4.1 Type A

For this example, a configuration was used as shown in Source Code 11.

```
1  ptxTransparentMode_RFParams_t transparent_mode_rf_params;
2  transparent_mode_rf_params.Tech = TM_RF_Tech_A;
3  transparent_mode_rf_params.TxRate = TM_RF_Bitrate_106;
4  transparent_mode_rf_params.RxRate = TM_RF_Bitrate_106;
5  transparent_mode_rf_params.NrTxBits = 7;
6  transparent_mode_rf_params.ResLimit = 1;
7  transparent_mode_rf_params.Flags =
   (uint8_t)(PTX_TRANSPARENT_MODE_FLAGS_TX_PARITY |
   PTX_TRANSPARENT_MODE_FLAGS_RX_PARITY);
```

Source Code 11. Type A Example Configuration

When looking at the example command captured in Source Code 11, you can see that it is modulated using a Modified Miller coding with Amplitude Shift Keying (ASK) 100% modulation with its X,Y, and Z pattern.

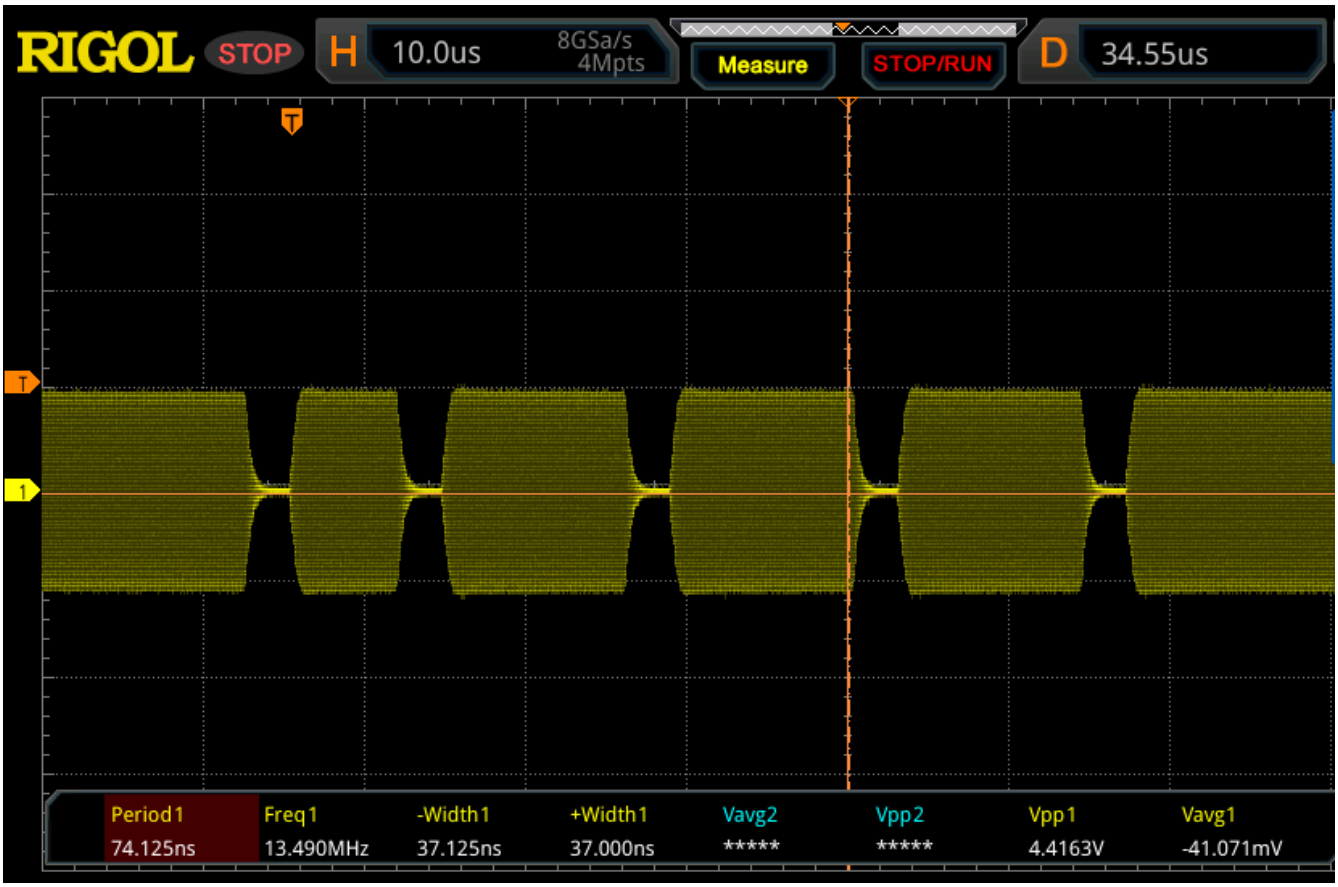

Figure 2. Type A Example Command 0x52

4.2 Type B

For this example, the configuration of Source Code 12 was used.

```
1 ptxTransparentMode_RFParams_t transparent_mode_rf_params;
2 transparent_mode_rf_params.Tech = TM_RF_Tech_B;
3 transparent_mode_rf_params.TxRate = TM_RF_Bitrate_106;
4 transparent_mode_rf_params.RxRate = TM_RF_Bitrate_106;
5 transparent_mode_rf_params.NrTxBits = 0;
6 transparent_mode_rf_params.ResLimit = 1;
7 transparent_mode_rf_params.Flags =
(uint8_t)(PTX_TRANSPARENT_MODE_FLAGS_TX_PARITY |
PTX_TRANSPARENT_MODE_FLAGS_RX_PARITY | PTX_TRANSPARENT_MODE_FLAGS_TX_CRC|
PTX_TRANSPARENT_MODE_FLAGS_RX_CRC);
```

Source Code 12. Type B Example Configuration

Compared to Type A in section 4.1, you can notice the far smaller ASK of 10% instead of 100% in Source Code 12. Therefore, it is also difficult to spot the Non-Return to Zero, L for level (NRZ-L) coding, and its H and L patterns.

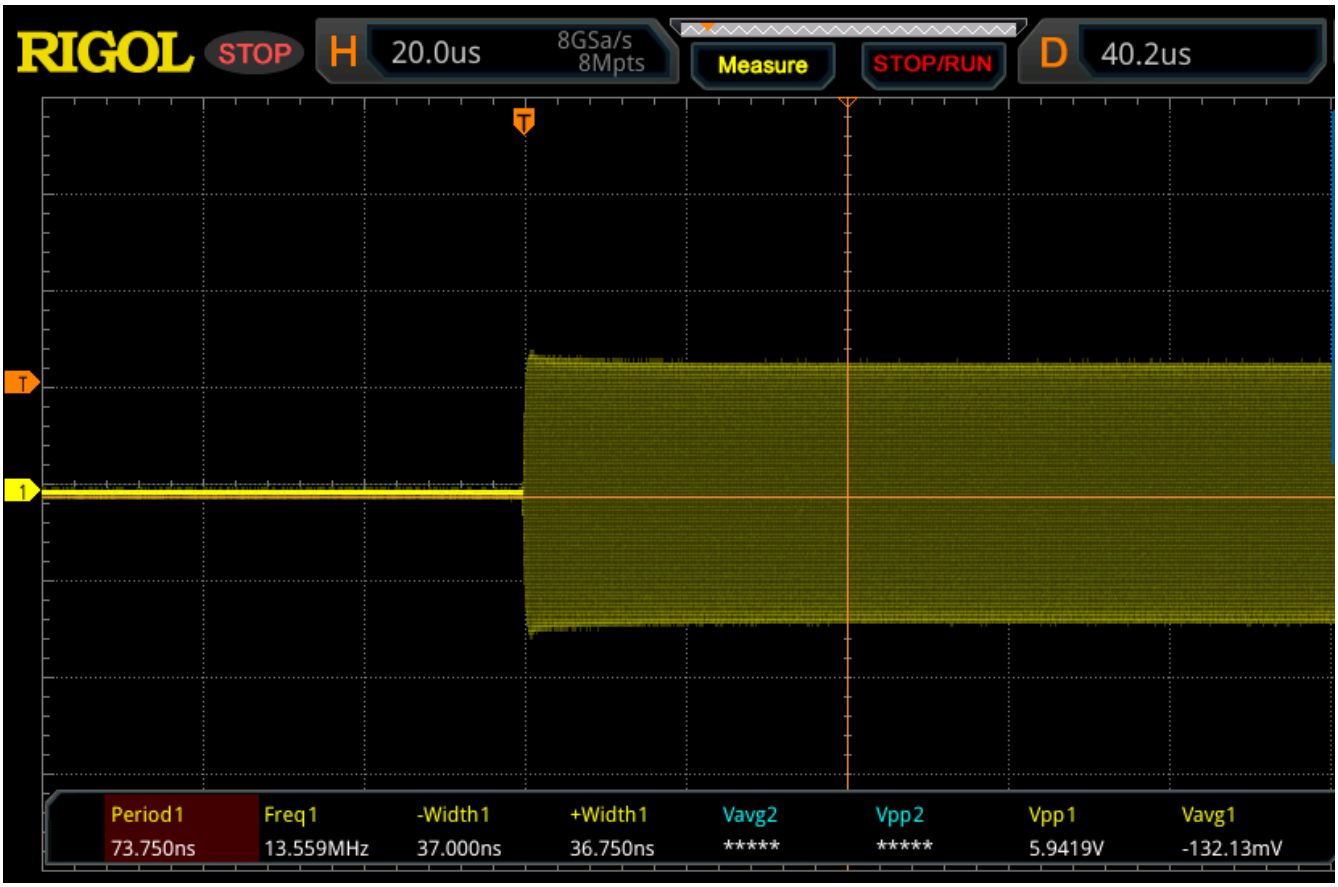

Figure 3. Type B Example Command 0x050000

4.3 Type F

For this example, the configuration of Source Code 13 was used.

```
ptxTransparentMode_RFParams_t transparent_mode_rf_params;
transparent_mode_rf_params.Tech = TM_RF_Tech_F;
transparent_mode_rf_params.TxRate = TM_RF_Bitrate_212;
transparent_mode_rf_params.RxRate = TM_RF_Bitrate_212;
transparent_mode_rf_params.NrTxBits = 0;
transparent_mode_rf_params.ResLimit = 1;
transparent_mode_rf_params.Flags =
(uint8_t)(PTX_TRANSPARENT_MODE_FLAGS_TX_PARITY |
PTX_TRANSPARENT_MODE_FLAGS_RX_PARITY | PTX_TRANSPARENT_MODE_FLAGS_TX_CRC|
PTX_TRANSPARENT_MODE_FLAGS_RX_CRC);
```

Source Code 13. Type F Example Configuration

Looking at Source Code 13, it looks similar to Source Code 12. This is because Type F also uses an ASK modulation of 10% like Type B. However, its Manchester coding with L and H patterns makes is different from Type B.

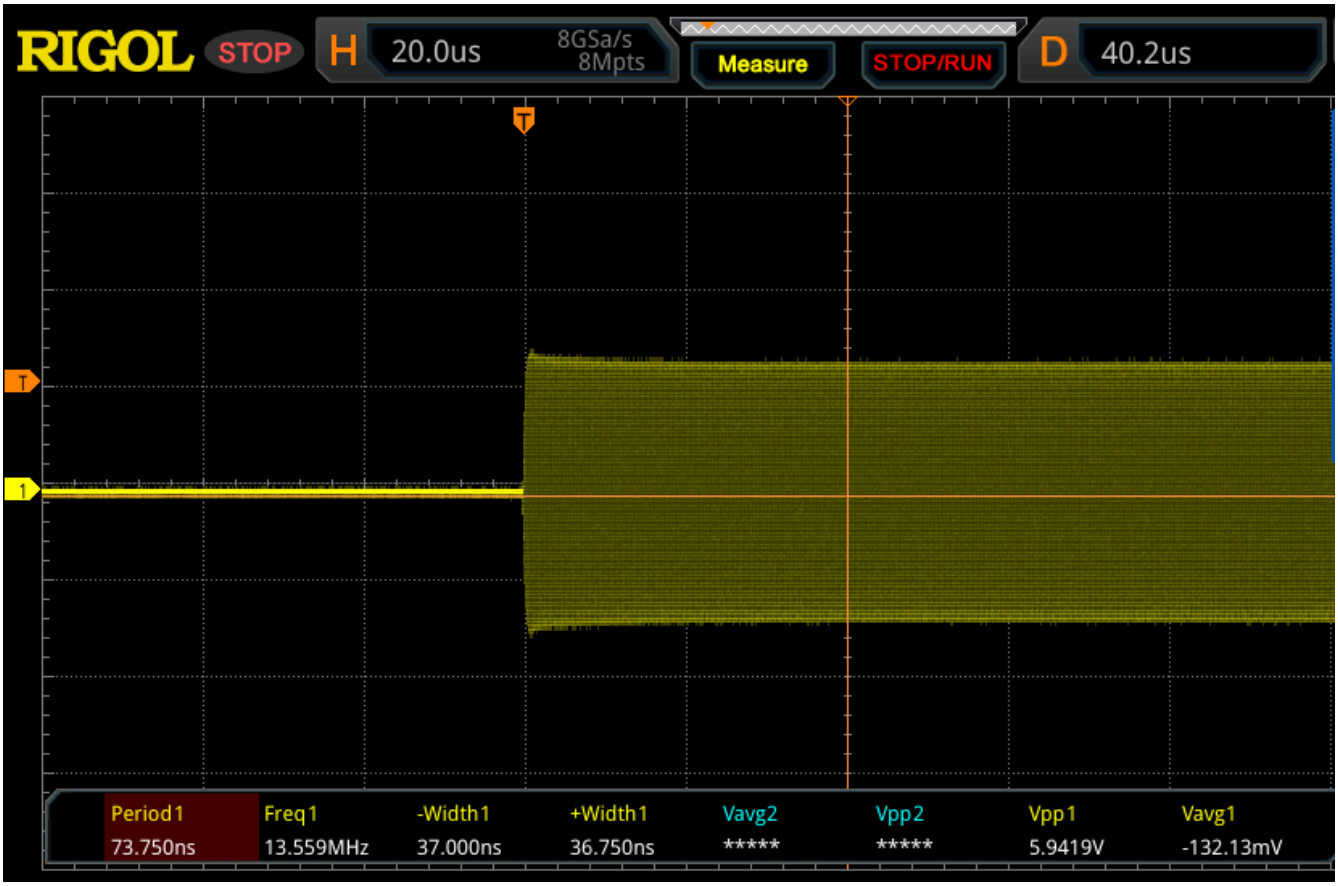

Figure 4. Type F Example Command 0x00FFFF0000

4.4 Type V

For this example, the configuration of Source Code 14 was used.

```
1 ptxTransparentMode_RFParams_t transparent_mode_rf_params;
2 transparent_mode_rf_params.Tech = TM_RF_Tech_V;
3 transparent_mode_rf_params.TxRate = TM_RF_Bitrate_26;
4 transparent_mode_rf_params.RxRate = TM_RF_Bitrate_26;
5 transparent_mode_rf_params.NrTxBits = 0;
6 transparent_mode_rf_params.ResLimit = 1;
7 transparent_mode_rf_params.Flags =
(uint8_t)(PTX_TRANSPARENT_MODE_FLAGS_TX_PARITY |
PTX_TRANSPARENT_MODE_FLAGS_RX_PARITY | PTX_TRANSPARENT_MODE_FLAGS_TX_CRC|
PTX_TRANSPARENT_MODE_FLAGS_RX_CRC);
```

Source Code 14. Type V Example Configuration

Type V's analog signal is modulated using pulse position coding with ASK modulation. In the Type V example, you can notice a similarity in the use of 100% ASK modulation to Type A.

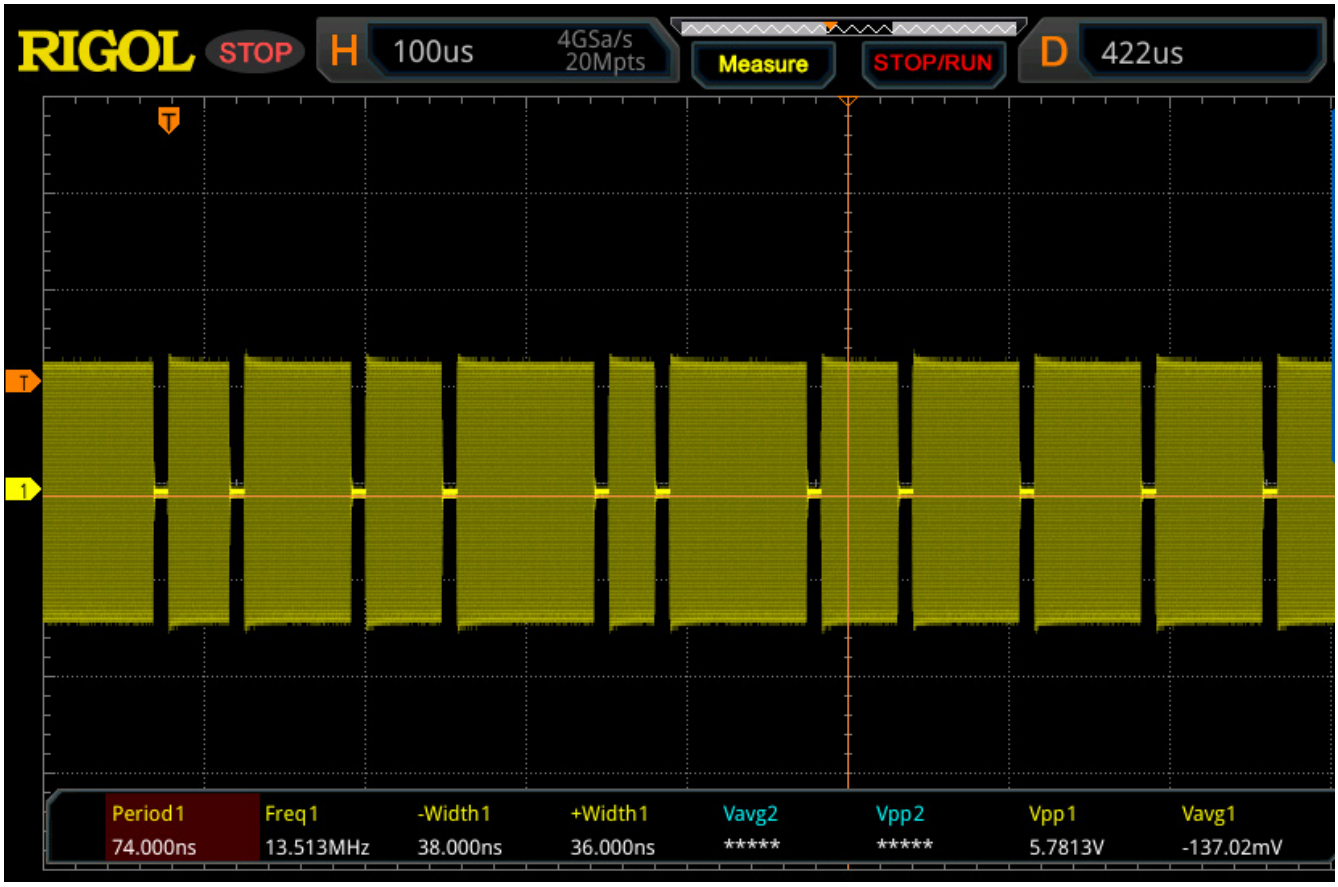

Figure 5. Type V Example Command 0x260100

5. Revision History

Revision	Date	Description
1.00	Jul 16, 2024	Initial release.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Jan 2024)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.