# AN13627

## The feature nIRQ notifier pin support of blhost for RT600

Rev. 0 — 22 April 2022 Application Note

Contents



# 1 Introduction

The RT600 is a family of dual-core microcontrollers featuring an Arm Cortex-M33 CPU combined with a Cadence Xtensa HiFi4 advanced Audio Digital Signal Processor. The RT600's bootloader supports In-System Programming or serial boot via the SPI/I2C peripheral, where the SPI / I2C peripheral serves as the SPI/I2C slave.

For more details on the RT600 MCU, see the documents available here.

The blhost application is used on a host to issue commands to an NXP platform running an implementation of the MCU bootloader.

For more details on the blhost application, download the blhost application `blhost_2.6.7` available here.

This application note introduces the implementation of the feature nIRQ pin, which the application processor (namely, the host) can use to improve the communication performance between the host and the RT600's bootloader via the SPI/I2C peripheral.

# 2 SPI/I2C transaction protocol

This section explains the general protocol for the packet transfers between the host and the RT600's bootloader.

## 2.1 Command with no data phase

The protocol for a command with no data phase contains:

- Command packet (from the host).
- Generic response command packet (to the host).

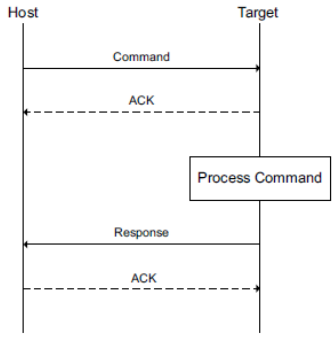


Figure 1. Command with no data phase

## 2.2 Command with incoming data phase

The protocol for a command with incoming data phase contains:

- Command packet (from the host) (the `kCommandFlag_HasDataPhase` set).

- Generic response command packet (to the host).
- Incoming data packets (from the host).
- Generic response command packet.

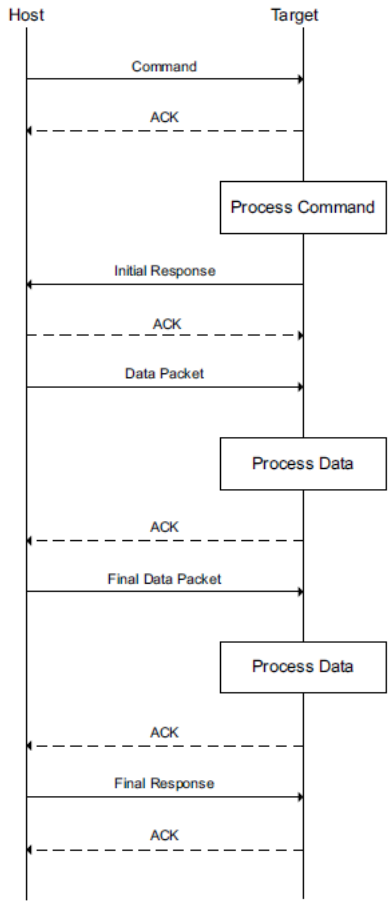


**Figure 2. Command with incoming data phase**

## 2.3 Command with outgoing data phase

The protocol for a command with an outgoing data phase contains:

- Command packet (from the host).
- ReadMemory Response command packet (to the host) (the `kCommandFlag_HasDataPhase` set).
- Outgoing data packets (to the host).
- Generic response command packet (to the host).

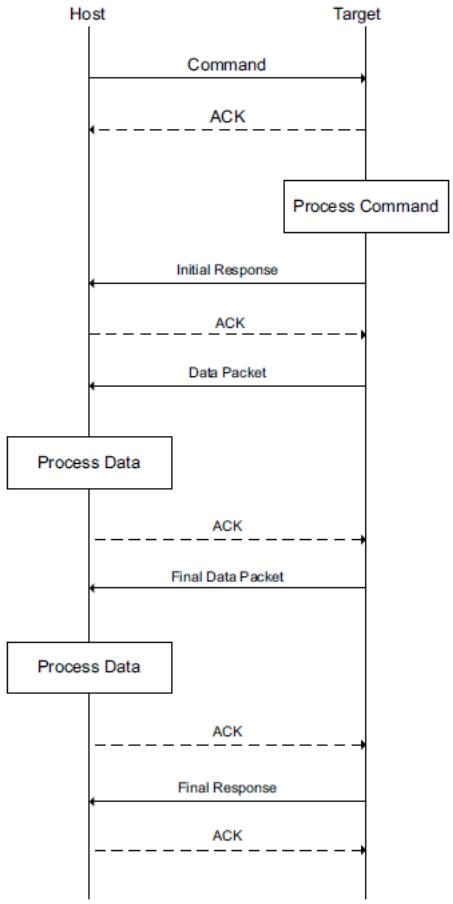


Figure 3. Command with outgoing data phase

## 2.4 The feature nIRQ notifier pin

Once the feature nIRQ notifier pin being enabled, the host waits until it catches a negative edge on the nIRQ notifier pin before reading any data from the RT600's bootloader. It must suspend operation until the nIRQ notifier pin is high before sending any commands/data to the RT600's bootloader. For each packet, the sent timeout time is the packet length (in byte) multiplied to 10 ms and the receive timeout time is the packet length (in byte) multiplied to 20 ms. It means that the host must send the packet to the RT600's bootloader before the sending packet timeout and receive the packet from to the RT600's bootloader before the receiving packet timeout.

# 3 The feature nIRQ notifier pin support of blhost

The blhost application, `blhost_2.6.7`, has the set-property command implemented to enable the feature nIRQ notifier pin.

```
set-property <tag> <value>[<memoryID>]
```

- tag:
  - 0x1c - The pin selected as IRQ notifier pin.
- value:
  - bit[31] - Enable nIRQ notifier pin, 0: disable, 1: enable.
  - bit[15:8] port.
  - bit [7:0] pin.

However, the blhost application, `blhost_2.6.7`, has not implemented the feature nIRQ notifier pin described in section The feature nIRQ notifier pin.

This section provides details on how the feature nIRQ notifier pin described in section The feature nIRQ notifier pin can be implemented.

## 3.1 Modifications to blhost.cpp

The `blhost.cpp` is in `\blhost_2.6.7\tools\blhost\src\`. The modifications are in bold below.

```
////////////////////////////////////////////////////////////////////////////////
// Variables
////////////////////////////////////////////////////////////////////////////////
/* set-property command execution */
extern char SetProperty_Flag;

int BlHost::run()
{
.
.
.
   try
    {
      if (m_cmdv.size())
      {
          // Check for any passed commands and validate command.
          cmd = Command::create(&m_cmdv);
          if (!cmd)
          {
              std::string msg = format_string("Error: invalid command or arguments '%s",
m_cmdv.at(0).c_str());
              string_vector_t::iterator it = m_cmdv.begin();
              for (++it; it != m_cmdv.end(); ++it)
              {
                msg.append(format_string(" %s", (*it).c_str()));
              }
              msg.append("'\n");
              throw std::runtime_error(msg);
            }
            progress = new Progress(displayProgress, NULL);
            cmd->registerProgress(progress);
          }
         //If the instruction is set-property, the flag is set
          if (cmd->getName() == "set-property")
          {
           SetProperty_Flag = 1;
          }
```

## 3.2 Modifications to Command.cpp

The `Command.cpp` is in `\blhost_2.6.7\src\blfwk\src\`. The modifications are in bold below.

```
////////////////////////////////////////////////////////////////////////////////
// Variables
////////////////////////////////////////////////////////////////////////////////
/* set-property command execution */
extern char SetProperty_Flag;
```

```
/* nIRQ notifier pin enabled */
extern char IrqNotifierPin_Flag;

void SetProperty::sendTo(Packetizer &device)
{
  blfwk::CommandPacket cmdPacket(kCommandTag_SetProperty, kCommandFlag_None, m_propertyTag,
m_propertyValue);
 if (SetProperty_Flag == 1)
  {
       if ((m_propertyTag == 0x1c)&&((m_propertyValue & 0x80000000) == 0x80000000))
       {
              IrqNotifierPin_Flag = 1;
       }
       else
       {
              SetProperty_Flag = 0;
       }
  }
```

## 3.3 Modifications to SerialPacketizer.cpp

The `SerialPacketizer.cpp` is in `\blhost_2.6.7\src\blfwk\src\`. The modifications are in bold below.

```
/* set-property command execution */
char SetProperty_Flag = 0;
/* nIRQ notifier pin enabled */
char IrqNotifierPin_Flag = 0;
////////////////////////////////////////////////////////////////////////////////
// Code
////////////////////////////////////////////////////////////////////////////////

status_t SerialPacketizer::serial_packet_write(const uint8_t *packet, uint32_t byteCount,
packet_type_t packetType)
{
.
.
.
    // If nIRQ notifier pin has been enabled, then waits for nIRQ notifier pin high here.
    if (SetProperty_Flag == 0) {
        // Waits for nIRQ notifier pin high.
        if(wait_for_high()!=kStatus_Success)
       {
            return kStatus_Timeout;
       }
    }
    // Send the framing data packet.
    status = m_peripheral->write((uint8_t *)framingPacket, sizeof(framing_data_packet_t) + byteCount);
    if (status != kStatus_Success)
    {
        return status;
    }
   // If current command is set-property to enable nIRQ notifier pin, then waits for nIRQ notifier
pin high here.
    if (SetProperty_Flag == 1) {
        SetProperty_Flag = 0;
        // Waits for nIRQ notifier pin high.
        if(wait_for_high()!=kStatus_Success)
        {
```

```
            return kStatus_Timeout;
        }
    }
    return wait_for_ack_packet();
}

status_t SerialPacketizer::serial_packet_send_sync(uint8_t framingPacketType)
{
    framing_sync_packet_t sync;
    sync.header.startByte = kFramingPacketStartByte;
    sync.header.packetType = framingPacketType;
    // Indicate last transaction was a write.
    m_serialContext.isBackToBackWrite = true;
   // Waits for nIRQ notifier pin high.
    if(wait_for_high()!=kStatus_Success)
    {
          return kStatus_Timeout;
    }

status_t SerialPacketizer::wait_for_ack_packet()
{
    framing_sync_packet_t sync;
    status_t status = kStatus_NoCommandResponse;
    do
    {
          // Waits for nIRQ notifier pin low.
        if(wait_for_low()!=kStatus_Success)
           {
              return kStatus_Timeout;
           }
.
.
.
          // Waits for nIRQ notifier pin high.
         if(wait_for_high()!=kStatus_Success)
           {
              return kStatus_Timeout;
           }
    return status;
}

status_t SerialPacketizer::read_data_packet(framing_data_packet_t *packet, uint8_t *data,
packet_type_t packetType)
{
         // Waits for nIRQ notifier pin low.
       if(wait_for_low()!=kStatus_Success)
          {
              return kStatus_Timeout;
          }
```

## 3.4 Implementation of wait_for_high() and wait_for_low()

The implementation of `wait_for_high()` and `wait_for_low()` is host-dependent.

This application note shows their implementation with RT600 as the host.

```
status_t wait_for_high(void)
{
    uint32_t duration = 0;
```

```
    if(IrqNotifierPin_Flag == 1)
    {
        uint32_t start = CTIMER_GetTimerCountValue(CTIMER0);
        while (duration < m_packetTimeoutMs)
        {
            if(GPIO_PinRead(GPIO, HOST_PORT, HOST_PIN)==1)
            {
                return kStatus_Success;
            }
            duration = (uint32_t)( CTIMER_GetTimerCountValue(CTIMER0) - start) / CLOCKS_PER_1MS;
        }
        return kStatus_Timeout;
    }
    return kStatus_Success;
}

status_t wait_for_low(void)
{
    uint32_t duration = 0;
    if(IrqNotifierPin_Flag == 1)
    {
        uint32_t start = CTIMER_GetTimerCountValue(CTIMER0);
        while (duration < m_packetTimeoutMs)
        {
            if(GPIO_PinRead(GPIO, HOST_PORT, HOST_PIN)==0)
            {
                return kStatus_Success;
            }
            duration = (uint32_t)( CTIMER_GetTimerCountValue(CTIMER0) - start) / CLOCKS_PER_1MS;
        }
        return kStatus_Timeout;
    }
    return kStatus_Success;
}
```

For more details on the SDK_2_11_0_EVK-MIMXRT685, refer to the software available here.

# 4 Conclusion

The application processor (namely, host) can use the feature nIRQ notifier pin to improve the communication performance between it and the RT600's bootloader, which can be enabled by the `SetProperty <tag> <value>` command.

# 5 Revision history

**Table 1. Revision history**

| Revision Number | Date | Substantive Changes |
|---|---|---|
| 0 | 22 April 2022 | Initial release |

# Legal information

## Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

## Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Terms and conditions of commercial sale** — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Suitability for use in non-automotive qualified products** — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

**Translations** — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

## Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

**NXP** — wordmark and logo are trademarks of NXP B.V.

**AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile** — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

**Airfast** — is a trademark of NXP B.V.

**Bluetooth** — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

**Cadence** — the Cadence logo, and the other Cadence marks found at www.cadence.com/go/trademarks are trademarks or registered trademarks of Cadence Design Systems, Inc. All rights reserved worldwide.

**CodeWarrior** — is a trademark of NXP B.V.

**ColdFire** — is a trademark of NXP B.V.

**ColdFire+** — is a trademark of NXP B.V.

**EdgeLock** — is a trademark of NXP B.V.

**EdgeScale** — is a trademark of NXP B.V.

**EdgeVerse** — is a trademark of NXP B.V.

**eIQ** — is a trademark of NXP B.V.

**FeliCa** — is a trademark of Sony Corporation.

**Freescale** — is a trademark of NXP B.V.

**HITAG** — is a trademark of NXP B.V.

**ICODE and I-CODE** — are trademarks of NXP B.V.

**Immersiv3D** — is a trademark of NXP B.V.

**I2C-bus** — logo is a trademark of NXP B.V.

**Kinetis** — is a trademark of NXP B.V.

**Layerscape** — is a trademark of NXP B.V.

**Mantis** — is a trademark of NXP B.V.

**MIFARE** — is a trademark of NXP B.V.

**MOBILEGT** — is a trademark of NXP B.V.

**NTAG** — is a trademark of NXP B.V.

**Processor Expert** — is a trademark of NXP B.V.

**QorIQ** — is a trademark of NXP B.V.

**SafeAssure** — is a trademark of NXP B.V.

**SafeAssure** — logo is a trademark of NXP B.V.

**StarCore** — is a trademark of NXP B.V.

**Synopsys** — Portions Copyright © 2021 Synopsys, Inc. Used with permission. All rights reserved.

**Tower** — is a trademark of NXP B.V.

**UCODE** — is a trademark of NXP B.V.

**VortiQa** — is a trademark of NXP B.V.



For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

**Date of release: 22 April 2022**
**Document identifier: AN13627**