RunBMC NPCM7mnx-Based Modules Microcontroller Implementation Guide

May 2019

REVISION RECORD

REVISION	RELEASE DATE	SUMMARY OF CHANGES
1.0	May 2019	First Revision.

Chapter 1
OVERVIEW

The Nuvoton RunBMC modules, which are based on the NPCM7mnx BMC, include the Atmel® ATtiny1634 microcontroller ("microcontroller"). This Implementation Guide describes the usage and implementation of the microcontroller on the RunBMC modules.

The microcontroller implements several functionalities:

1. Main functions:
   - SPI Flash power-cycle
   - I2C GPI expender emulator (connected to GPI0-7 pins on the RunBMC SO-DIMM connector)
   - I2C ADC emulator (connected to ADC8-15 pins on the RunBMC SO-DIMM connector)

2. Optional and debug functions:
   - I2C 64 KB EEPROM emulator
   - I2C SRAM emulator
   - Watchdog emulator
   - UART TX emulator

The block diagram below is an abstract description of the microcontroller functionalities.

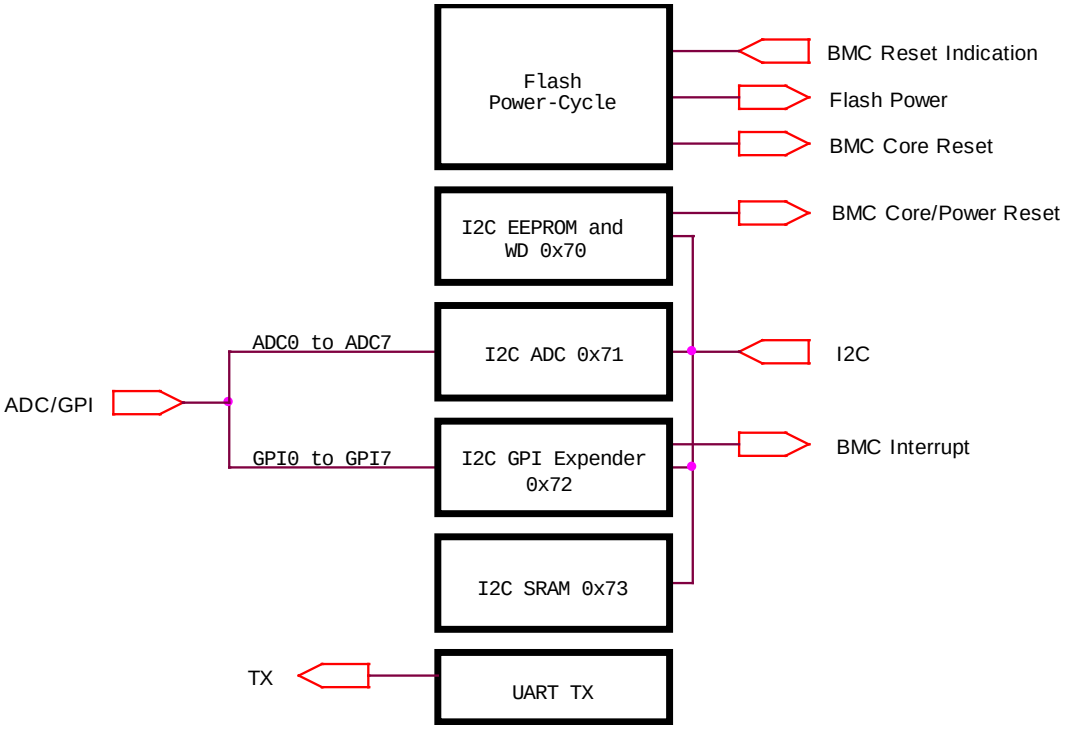


Figure 1: Microcontroller Functionalities

1.1 ATtiny1634 Microcontroller Main Features

The main microcontroller features used in the RunBMC modules are:

- 12-channel, 10-bit ADC (8 channels used)
- Slave I2C interface
- External interrupt sources (INT0 used)
- In-system programming via external serial programming
- 8 MHz internal clock
- 16 KB flash (~8 KB used)
- 1 KB RAM (~128 bytes used)
- 256B EEPROM (optional)

1.2 Microcontroller Connectivity

The diagram (simplified) below shows the connectivity of the microcontroller, NPCM7mnx BMC, firmware SPI flash and SO-DIMM connector in the RunBMC modules.

For more details, see the accompanying Nuvoton *RunBMC Reference Schematics* document.

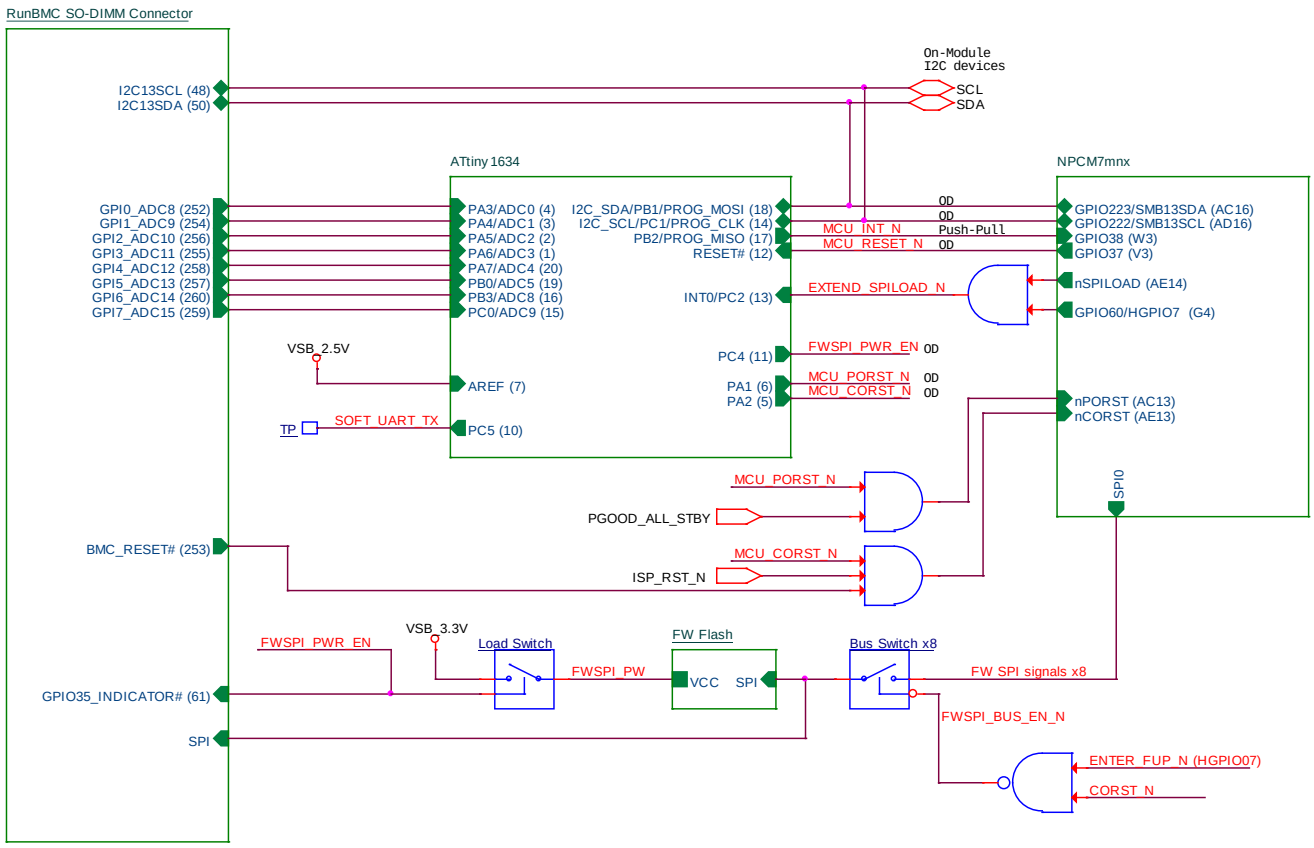


Figure 2: ATtiny1634 Microcontroller Connectivity Diagram

1.3 Microcontroller I2C Slave Interface

The microcontroller is programmed to emulate four slave devices on I2C base address 0x70. Before programming, the base address can be changed in the binary source file; see Section 1.4.3.

While the microcontroller is in an I2C transaction (between START, with matched addresses, and STOP conditions), a timeout counter is enabled and set to 100 msec. The counter is restarted each time the microcontroller sends or receives an I2C byte. When a timeout occurs, the microcontroller restarts its I2C module and releases its I2C pins.

1.4 In-System Programming of the Microcontroller

The microcontroller supports an external Serial Programming interface for reading, erasing and programming the internal flash, EEPROM, fuses and lock bits. For more details about microcontroller serial programming, see the *ATtiny1634 Datasheet*, Section 2.3.

Note: The RunBMC modules are designed so that the NPCM7mnx can power up to UBOOT and Linux even when the microcontroller is in reset or its flash is empty (i.e., no firmware---the default factory). Note that when the microcontroller is in reset or its flash is empty, all its I/Os are kept in TRI-STATE.

1.4.1 External Serial Programming Interface

Serial programming interface signals are connected to the NPCM7mnx to allow microcontroller in-system programming using programming tool runs from UBOOT. The table below describes the serial programming signal mapping between the four microcontroller pins and the NPCM7mnx signals.

ATtiny1634 Pin	NPCM7mnx Pin	Pin Usage in Programing Mode	Pin Usage in Normal Mode
PC1	GPIO222	Serial Clock (SCLK)	I2C SCL CH13
PB1	GPIO223	Serial Data In (MOSI)	I2C SDA CH13
PB2	GPIO38	Serial Data Out (MISO)	INT# (Interrupt)
RESET#	GPIO37	RESET#	RESET#

The microcontroller enters Serial Programming mode when the input RESET# pin goes low while the input SCLK pin is low. In addition, the NPCM7mnx GPIO37 output type is set to open-drain and there is an external 10K pull-up resistor on the RESET# signal.

Note: For added security, while in UBOOT and just before continuing to Linux, it is recommended, from the NPCM7mnx side, to set GPIO37 to output high and then lock the pin state against further changes by setting bit 5 in GP1SPLCK register. This prevents programming or resetting the microcontroller while in Linux. GPIO38, GPIO223 and GPIO222 can also be locked in this way.

Note that although SCLK/SCL and MOSI/SDA signals are shared with other I2C devices, the I2C devices will **not** interfere with in-system programming. This is because while in programming mode, the MOSI/SDA pin changes **only** while the SCLK/SCL pin is low (i.e., an I2C START/STOP condition cannot be detected by the I2C devices, which therefore stay inactive).

1.4.2 Programming Clock Frequency

The SCLK frequency is limited to ¼ the microcontroller's CPU clock frequency. Factory default values of the CKSEL and CKDIV8 fuses set the CPU clock to an internal 8 MHz oscillator divided by 8 (i.e., resulting in a 1 MHz CPU clock). Therefore, SCLK is limited to 250 KHz. In addition, since the SCLK and MOSI signals are shared with I2C Channel 13, which is connected to several I2C devices (on or off system), the SCLK frequency is limited by the I2C Channel 13 bus load and pull-up values.

Note: It is **recommended** to use open-drain output type for SCLK and MOSI signals and an SCLK frequency of up to 100 KHz.

1.4.3 Binary File Information

The binary file includes date, time, revision and configuration information, which can be used by the programmer utility to identify the file revision and set the configuration before programming.

File Offset	Byte Length	Description
0x70	16	Null-terminated header string: "**Nuvoton_RunBMC**".
0x80	16	Null-terminated date string, auto generated using the **__DATE__** macro.
0x90	16	Null-terminated time string, auto generated using the **__TIME__** macro.
0xA0	1	Hex value for minor revision.
0xA1	1	Hex value for major revision.
0xA2	1	Hex value of theI2C 7-bit base address for the four emulated devices. This value can be updated before programming to change the I2C base address. The default is 0x70.
0xA3	1	Hex value for UART TX output on port C. The default is 0x05 for PC5; it can be changed to PC0 by writing a value of 0x00.
0xA4	12	Reserved.

File Example:

ATtiny1634_FW.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   0C 94 5E 03 0C 94 6A 0A 0C 94 7B 03 0C 94 7B 03  .”^..”j..”{..”{.
00000010   0C 94 7B 03 0C 94 39 0A 0C 94 7B 03 0C 94 7B 03  .”{..”9..”{..”{.
00000020   0C 94 7B 03 0C 94 7B 03 0C 94 A8 0B 0C 94 7B 03  .”{..”{..”¨..”{.
00000030   0C 94 7B 03 0C 94 7B 03 0C 94 7B 03 0C 94 7B 03  .”{..”{..”{..”{.
00000040   0C 94 7B 03 0C 94 7B 03 0C 94 7B 03 0C 94 7B 03  .”{..”{..”{..”{.
00000050   0C 94 7B 03 0C 94 7B 03 0C 94 7B 03 0C 94 7B 03  .”{..”{..”{..”{.
00000060   0C 94 7B 03 0C 94 33 08 0C 94 7B 03 0C 94 7B 03  .”{..”3..”{..”{.
00000070   4E 75 76 6F 74 6F 6E 5F 52 75 6E 42 4D 43 00 00  Nuvoton_RunBMC..
00000080   41 70 72 20 20 31 20 32 30 31 39 00 00 00 00 00  Apr  1 2019.....
00000090   31 36 3A 33 32 3A 34 30 00 00 00 00 00 00 00 00  16:32:40........
000000A0   03 00 70 05 00 00 00 00 00 00 00 00 00 00 00 00  ..p.............
```

1.4.4 Programmer Utility

Nuvoton provides the ATtiny1634 programmer utility running from UBOOT.

Chapter 2
MICROCONTROLLER FUNCTIONALITY

2.1 Flash Device Power-Cycle

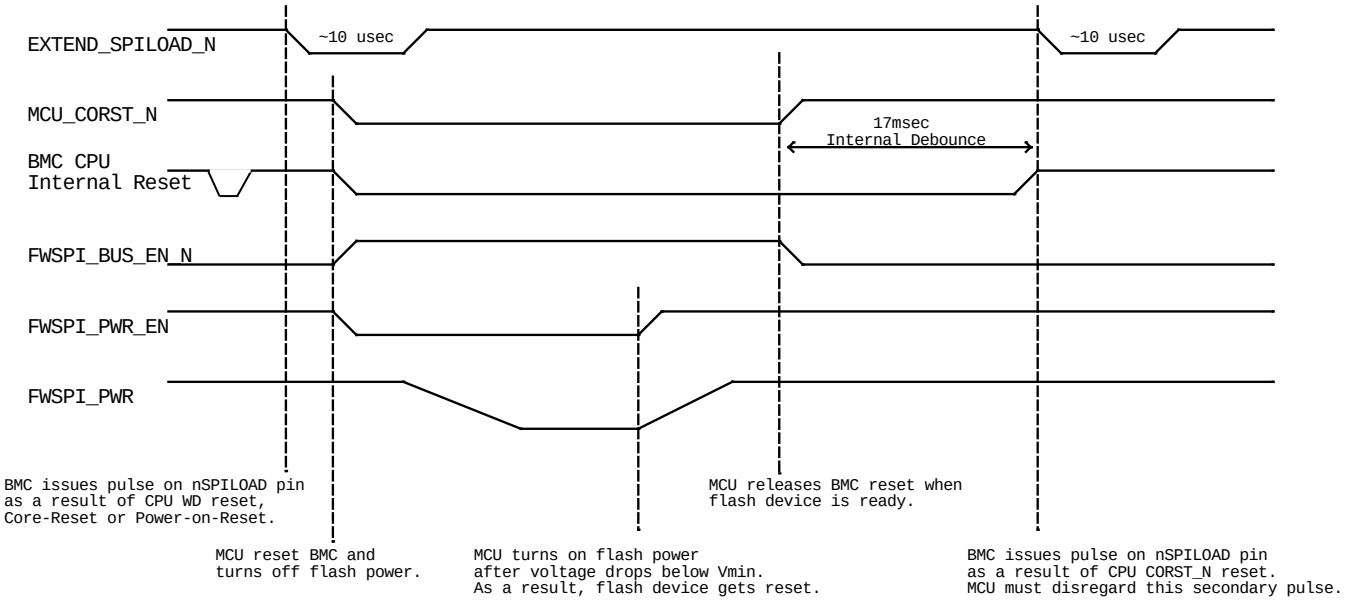
It is of utmost importance to reset the flash device connected to nSPI0CS0 when the NPCM7mnx CPU is reset (e.g., due to a Watchdog reset). This is most essential with flash devices above 16 MB. The NPCM7mnx ROM code will fail to read the boot block from the flash device if the mode is incorrect (e.g., if the flash device is configured as 32-bit address mode instead of 24-bit address mode) or if the device is not ready (e.g., is in the middle of an erase process).

The NPCM7mnx includes the nSPILOAD output reset pin, which can be used to reset the SPI flash device. When the NPCM7mnx CPU is reset (e.g., due to a Watchdog reset, Software reset or nCORST), a 0.5 µsec low-level pulse is generated on the nSPILOAD pin. The nSPILOAD pin is push-pull output and powered by the VSB33 power supply.

In the RunBMC modules, the 'flash power-cycle' method is used to reset the flash. To avoid back-drive from NPCM7mnx SPI signals (e.g., nSPI0CS0 pin is high on NPCM7mnx reset) to a powered OFF flash, a bus switch is used to disconnect SPI signals from the NPCM7mnx when the flash is powered OFF.

Since there is a risk that the microcontroller may fail to detect an nSPILOAD short pulse, an RC pulse extender circuit was added. Note that using a 100 KΩ pull-up resistor together with a 1 nF capacitor results in an extended pulse of ~10 µsec.

Flash Reset using Power Cycle Method



The Microcontroller Implements the following procedure:

This procedure is based on the timing diagram above.

1. Wait for a NCPM7mnx reset event (detected by a falling edge on the EXTEND_SPILOAD_N signal).
2. Following the reset event, hold the NCPM7mnx in reset by setting MCU_CORST_N signal low. Note that SPI Bus Switch Enable signal (FWSPI_BUS_EN_N) is connected to the CORST_N signal. Therefore, when the microcontroller resets the NPCM7mnx, the SPI bus switch is disabled.
3. Turn OFF the flash device power supply by setting the PWSPI_PWR_EN signal to low.
4. Wait 5 msec. This should be enough time for the flash device power to discharge.
5. Turn ON the flash device power by setting the PWSPI_PWR_EN signal to high. Wait until the signal goes high (due to an on-module pull-up resistor). Note that the PWSPI_PWR_EN signal is an open-drain output type routed to the motherboard. The motherboard can use this feature to extend the power-off time by keeping the PWSPI_PWR_EN signal low.
6. Wait 5 msec. This should be enough time for flash device power to rise up to normal voltage.
7. Release the NPCM7mnx from reset by setting the MCU_CORST_N signal high. Note that MCU_CORST_N is an Open-Drain output signal.
8. Disregard the second pulse on the EXTEND_SPILOAD_N signal (i.e., wait until an NPCM7mnx reset event occurs, indicated by a falling edge on the EXTEND_SPILOAD_N signal).
9. Return to Step 1.

2.1.1 Enter FUP Mode From the Host

In the RunBMC modules, the Host can place the NPCM7mnx in Flash UART Programming (FUP) mode. In FUP mode, when the STRAP(10-9) value is set to 11b, the flash can be programmed using Host serial port 2 (SP2), which is mapped to LPC/eSPI Logical Device 2.

Entering FUP mode by the Host is done by setting the Host HGPIO7 pin low. This causes the EXTEND_SPILOAD_N signal to go low and the microcontroller to start the flash device power-cycle procedure, described above. In addition, keeping the HGPIO7 pin low keeps the SPI bus switch open. When a flash device power-cycle is completed and the NPCM7mnx Core reset is released, the NPCM7mnx ROM code starts running, fails to read the boot block from the flash and therefore enters FUP mode.

Once the NPCM7mnx is in FUP mode, the Host sets HGPIO7 high to close the SPI bus switch and then uses SP2 to communicate with the FUP monitoring code and program the flash.

For more details on FUP mode, see *NPCM7mnx Datasheet, Appendix A*: "Flash UART Programming Protocol". Note that Nuvoton provides tools for flash programming using FUP mode.

To Enter FUP Mode from the Host, Do the Following:

1. From the NPCM7mnx, set the GPIO60/HGPIO7 mux to HGPIO7 by setting HG7SEL bit in MFSEL2 register high. The default mux state after Power-Up reset is GPIO60. MFSEL2 register is reset only at Power-Up reset (i.e., when PWRGD_PS goes low).
   It is recommended to add this task to the beginning of the boot block code.
2. From the Host side,
   a. Configure the Host General-Purpose I/O Ports (HGPIO) module on LPC/eSPI Logical Device 7.
   b. You may keep the default output open-drain type (OUTTYPE bit in HGPCFG1 configuration register).
   c. Set HGPIO7 output low by setting high bit 7 in HGPDO register.
3. Once the NPCM7mnx is in FUP mode, set HGPIO7 output back to high.
4. Use the SP2 module in LPC/eSPI Logical Device 2 to program the flash.

Minimum Requirements:

1. At least the first section of the boot block must execute once after Power-Up reset. In this section the code must:
   a. For LPC, clear HOSTWAIT bit in SHM SMC_CTL register to enable LPC.
   b. For LPC/eSPI, configure GPIO60/HGPIO7 mux to HGPIO7.
2. The STRAP(10-9) value is set to 11b; therefore, when ROM code enters FUP mode it communicates using SP2.
3. If the NPXM7mnx 'Secure Boot' feature is enabled in the ROM code (i.e., FUSTRAP.oSECBOOT fuse is set), then the 'Halt on Failure' feature should **not** be enabled (FUSTRAP.oHLTOF fuse is cleared).

Disabling Entering FUP Mode from the Host:

For security purposes, entering FUP mode from the Host may need to be disabled. To disable it, perform one of the following:

- From the RunBMC module, remove the physical connection between HGPIO7 pin (ENTER_FUP_N signal) and the EXTEND_SPILOAD_N and FWSPI_BUS_EN_N signals.
- Change the microcontroller code so that the it waits until the EXTEND_SPILOAD_N signal goes high before continuing with the flash device power-cycle procedure. This keeps the NPCM7mnx in reset until the HGPIO7 signal goes back high and therefore, the NPCM7mnx will boot normally.
- From the NPCM7mnx, make sure that GPIO60 (muxed with HGPIO7) is selected (that is, HGPIO7 is not selected) and then set RLOCKR.SEL2LK bit to lock MFSEL2 register.
  Note: Setting RLOCKR.SEL2LK bit locks all MFSEL2 register bits.
- From the Host BIOS, configure HGPIO7 to output high or input and then lock the pin state by setting LOCKCFP bit in HGPCFG1 register.
- In NPXM7mnx fuse programming, enable 'Halt on Failure' by programming the FUSTRAP.oHLTOF fuse.

2.2 I2C GPI Expender Emulator

The microcontroller emulates an I2C expender with eight GPI signals, which are connected to GPI0-7 pins, respectively. Note that these signals are connected to ADC8-15 signals. See Figure 2 for details.

2.2.1 I2C GPI Expender Features

The main features are:

- I2C 7-bit address 0x72.
- One port, 8 inputs.
  Note: The microcontroller pins support input and output even though output is not supported by the RunBMC standard.
- Input ports are 3.3V.
- Maskable, latching transition detection with interrupt output to the NPCM7mnx.
- Port value, transition flags and INT# signal are updated every 1 msec.
- Software interface compatible to the MAX7319 I2C Port Expander.

2.2.2 Software Interface

I2C Read Transactions:

1. On Address acknowledge:
   - Releases the INT# output pin.
2. On Byte 0:
   - Samples the current port value and update the transition flags.
   - Stores the current transition flags and reset them.
   - Returns the port value.
3. On Byte 1:
   - Returns values of the stored transition flags.
4. On continuous reads, goes to step 2.

I2C Write Transactions:

1. On Address acknowledge:
   - Releases the INT# output signal.
2. On Byte 0:
   - Updates the transition pin interrupt mask register.
3. On continuous writes, goes to step 2.

Notes:

- During a read or write transaction, the INT# output signal stays high regardless of the transition flags.
- The INT# output signal is updated as needed at the end of the transaction.

UBOOT Console Example:

1. Enable interrupt INT# on a GPI7 transition and read port value and transition flags. On the test board, set GPI7 and GPI6 high:

```
i2c md 0x72 80.1 2
0080: c0 00    ..
```

2. On the test board, set GPI7 low. INT# is asserted. Then, when reading the GPI port value, INT# will be deasserted:

```
i2c md 0x72 80.1 8
0080: 40 80 40 00 40 00 40 00    @.@.@.@.
```

3. On the test board, set GPI7 high. INT# is asserted. Then, when reading the GPI port value, INT# will be deasserted:

```
i2c md 0x72 80.1 8
0080: c0 80 c0 00 c0 00 c0 00    ........
```

2.3 I2C ADC Emulator

The microcontroller emulates the I2C ADC using eight channels: analog inputs (ADC0 to ADC7) are connected to ADC8 to ADC15 signals on the RunBMC connector. Note that these signals are shared with GPI0-7 signals. See Figure 2 for details.

2.3.1 I2C ADC Features

The main features are:

- I2C 7-bit address 0x71.
- 8-bit, 8-channel ADC with internal analog MUX.
- 8-channel Single-Ended (SE) mode or 4-channels Differential (DIFF) mode.
  Note: DIFF mode is emulated by two SE measurements.
- Support reference voltage options:
  - 3.3V from the microcontroller VCC pin
  - 2.5V from the microcontroller AREF pin
- The software interface is compatible with both the ADS7830 (Texas Instruments) and CD9830 (ON Semiconductor®).

For ADC analog specifications, see the *ATtiny1634 Datasheet*.

2.3.2 Software Interface

I2C Read Transactions:

1. On Byte 0:
    - Sets reference voltage and analog MUX for the selected channel.
    - Starts conversion (this takes ~3.125 µsec for SE mode and ~6.25 µsec for DIFF mode).
    - Returns the measurement value.
2. On continuous reads, goes to Step 1.

I2C Write Transactions:

1. On Byte 0:
    - Updates the ADC Control register.
2. On continuous writes, responds with NACK.

ADC Control Register:

Bit	7	6	5	4	3	2	1	0
Name	**Mode**	**Input Channel**			**Reference Voltage**	**Reserved**		

Bit	Type	Description
7	RW	**Mode.** 0: Differential Mode (DIFF). Note that DIFF mode is emulated by two SE measurements. 1: Single-Ended Mode (SE).
6-4	RW	**Input Channel.** Channels are according to the mode selected (SE or DIFF), as follows:
3	RW	**Reference Voltage.** 0: 3.3V, from the VCC power supply pin. 1: 2.5V input to the AREF pin.
2-0	RO	**Reserved.** Keep value 0.

Input Channel	SE Mode	DIFF Mode
0 0 0	ADC0	ADC0 – ADC1
0 0 1	ADC2	ADC2 – ADC3
0 1 0	ADC4	ADC4 – ADC5
0 1 1	ADC6	ADC6 – ADC7
1 0 0	ADC1	ADC1 – ADC0
1 0 1	ADC3	ADC3 – ADC2
1 1 0	ADC5	ADC4 – ADC5
1 1 1	ADC7	ADC7 – ADC6

UBOOT Console Example:

1. Set mode to SE, reference to 3.3V, channel 7; read eight times
   On the board: connect the ADC15 pin to the 2.5V source.

```
i2c md 0x71 f0.1 8
00f0: bd bd bd bd bd bd bd bd    ........
```

   Note: 0xbd ➔ (189/256) * 3.3 = 2.436V

2. Set mode to SE, reference to 2.5V, channel 7; read x8 times

```
i2c md 0x71 f8.1 8
00f8: ff ff ff ff ff ff ff ff    ........
```

3. Set mode to DIFF, reference to 3.3V, channel 7; read x8 times.
   On the board: connect the ADC15 pin to the 2.5V source and the ADC14 pin to the 1.8V source; ADC15-ADC14 = 0.7V;

```
i2c md 0x71 70.1 8
0070: 35 35 35 35 35 35 35 35    55555555
```

   Note: 0x35 ➔ (53/256) * 3.3 = 0.683V

4. Set mode to DIFF, reference to 3.3V, channel 3; read x8 times.
   On the board: connect the ADC15 pin to the 2.5V source and the ADC14 pin to the 1.8V source; ADC14-ADC15 = (-) 0.7V;

```
i2c md 0x71 30.1 8
0030: cb cb cb cb cb cb cb cb    ........
```

   Note: 0xcb ➔ (203/256) * 3.3 = 2.616V (note that this value is incorrect because the current implementation does not support negative results.)

2.4 I2C SRAM Emulator

The microcontroller emulates a I2C 64 KB SRAM. There is no actual 64 KB SRAM: only 512 bytes are used. Attempting to access more than 512 bytes causes a wraparound. Note that this module is used for debug.

2.4.1 Software Interface

The software interface is compatible with standard I2C 64 KB FRAM/SRAM devices.

Writing Memory:

<I2C Address + **W**> <Address MSB> <Address LSB> <Data 0>…. <Data n>

Reading Memory:

1. Optional step (to set a new internal SRAM address):
   <I2C Address + **W**> <Address MSB> <Address LSB>
2. Read the Data:
   <I2C Address + **R**> <Data 0>…. <Data n>

UBOOT Console Example:

To address 0x140 write a value of 0x12 and read back 16 bytes.

```
i2c mw 0x73 0x0140.2 0x12
i2c md 0x73 0x0140.2 0x10

0140: 12 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
```

2.5 I2C EEPROM & WD Emulator

The microcontroller emulates a 64 KB I2C EEPROM device. There is no actual 64 KB EEPROM: this emulator is used for easy access (memory mapped) to other areas and emulated modules. Note that this module is used for debug.

2.5.1 Software Interface

The software interface is compatible with standard I2C 64 KB EEPROM devices, however, with the following limitations:

- Page Write is not supported.
- Direct 'Byte Write' is fully supported in some areas; in others it is supported only after a 'write-enable' sequence.
- Some areas are read-only.

EEPROM Memory Map:

Address Space	Type	Size	Module	Note
0x0000...0x007F	R/W	128 bytes	EEPROM for generic usage	Write is via 'write-enable' sequence
0x0080...0x00FF	RO	128 bytes	EEPROM for event logging	
0x0100...0x013F	RO	64 bytes	Firmware information as data, time and version	Same mapping as the binary file describe in Section 1.4.3
0x1000...0x14FF	RO	1280 bytes	Mirror of entire microcontroller SRAM	Data Memory (SRAM) and Register Files
0x3000...0x3004	R/W	5 bytes	Watchdog Reset module	Write is via 'write-enable' sequence
0x4000...0x7FFF	RO	16 KB	Mirror of entire microcontroller Flash	
0x8000...0x8003	R/W	4 bytes	'write-enable' sequence	
Other	Reserved; returns a value of 0xEE			

UBOOT Console Example:

Read firmware information from address 0x100.

```
i2c md 0x70 0x0100.2 0x40

0100: 4e 75 76 6f 74 6f 6e 5f 52 75 6e 42 4d 43 00 00    Nuvoton_RunBMC..
0110: 41 70 72 20 20 31 20 32 30 31 39 00 00 00 00 00    Apr  1 2019.....
0120: 31 35 3a 31 36 3a 31 31 00 00 00 00 00 00 00 00    15:16:11........
0130: 03 00 70 05 00 00 00 00 00 00 00 00 00 00 00 00    ..p.............
```

2.5.2 Write-Enable Sequence

Writes to some areas in the EEPROM emulator are only permitted via a write-enable sequence .To write 8-bit data to a 16-bit address, do the following

1. Write bits 15-8 of the target address to EEPROM address 0x8000.
2. Write bits 7-0 of the target address to EEPROM address 0x8001.
3. Write bits 7-0 of the target data to EEPROM address 0x8002.
4. Write bits 7-0 of the target data to the target address.

Notes:

- Repeat the above sequence for each byte write.
- A Write-Enable sequence is reset either after the last write (Step 4) or after an illegal write (i.e., a write to a read-only area).
- Illegal write transactions are answered with NACK.

UBOOT Console Example:

Write byte with a data value of 0x12 to EEPROM address 0x0040 , as follows:

```
i2c mw 0x70 0x8000.2 0x00 1      // write-enable sequence : Address MSB
i2c mw 0x70 0x8001.2 0x40 1      // write-enable sequence : Address LSB
i2c mw 0x70 0x8002.2 0x12 1      // write-enable sequence : Data
i2c md 0x70 0x8000.2 3           // optional, read 'write-enable' registers.
i2c mw 0x70 0x0040.2 0x12 1      // write to target
i2c md 0x70 0.2 100              // optional, read all EEPROM from 0x00 to 0xFF
```

2.5.3 Watchdog Reset

The microcontroller emulates a Watchdog reset module mapped on the EEPROM address as shown in the Memory Map table in Section 2.5.1. The Watchdog reset can trigger either a NPCM7mnx Core reset (via the nCORST pin) or NPCM7mnx Power-Up reset (via to nPORST pin) depending to the configuration (see below). Note that the NPCM7mnx includes several built-in watchdog modules that can generate a Core reset but none of them can generate a Power-Up reset.

WARNING:
Power-Up reset resets the Core and Host domains, and the DDR interface. If a Power-Up reset is performed, a Host that uses LPC/eSPI or PCIe interfaces may fail or freeze.

The Watchdog module is disabled on microcontroller power-up. However once the module is configured and enabled it cannot be disabled or changed. The Watchdog module is also disabled after a countdown to zero.

WD Control Register:

The first write to this register enables the Watchdog module. Further writes touch the Watchdog. Once any value other than 0 is written, the value of this register cannot be changed.

Address: 0x3000

Bit	7	6	5	4	3	2	1	0
Name	**TIME_OUT**				**Reserved**		**RESET_TYPE**	

Bit	Type	Description
7-4	R/W	**TIME_OUT.** Timeout in sec. Calculate as follows: 2 ^ **TIME_OUT.** 0: 1 sec 1: 2 sec 2: 4 sec 3: 8 sec … 14: 16, 384 sec 15: 32,768 sec (~9 hours)
3-2	R/W	**Reserved.**
1-0	R/W	**RESET_TYPE.** 0 0: No reset will be issued on a Watchdog timeout. 0 1: NPCM7mnx Core-Reset will be issued on a Watchdog timeout. 1 0: NPCM7mnx Power-Up reset will be issued on a Watchdog timeout. 1 1: Reserved.

WD Counter Register:

The Watchdog includes an internal 32-bit counter that counts down to zero (i.e., timeout, causing a reset). This register is read from address 0x3001 to 0x3004.

- Address 0x3001: Returns bits 7-0 of the 32-bit counter.
- Address 0x3002: Returns bits 15-8 of the 32-bit counter.
- Address 0x3003: Returns bits 23-16 of the 32-bit counter.
- Address 0x3004: Returns bits 31-24 of the 32-bit counter.

UBOOT Console Example:

Enable (& touch) the Watchdog, set the timeout to 32 seconds and issue an NPCM7mnx Power-Up reset on timeout.

```
i2c mw 0x70 0x8000.2 0x30 1    // write-enable sequence: Address MSB
i2c mw 0x70 0x8001.2 0x00 1    // write-enable sequence: Address LSB.
i2c mw 0x70 0x8002.2 0x52 1    // write-enable sequence: Data
i2c md 0x70 0x8000.2 3         // optional, read the 'write-enable' registers
i2c mw 0x70 0x3000.2 0x52 1    // write to target WD Control register
i2c md 0x70 0x3000.2 5         // optional, read WD Control register and counter
```

2.6 UART TX Emulator

The microcontroller emulates a UART TX module. The UART is used for microcontroller debug messages. By default the UART TX output pin is set to PC5 pin, which is connected to a test pad on the RunBMC module. This output pin can be changed to PC0, which is routed to the motherboard as the GPI7/ADC15 pin. See Section 1.4.3 for more information on how to set the UART TX output pin.

UART settings: 115.2 Kb, 1 stop bit, 1 start bit.

Example log:

```
 ********************
 Hello from ATtiny1634
 ********************

> Build Date: Apr  1 2019; 15:16:11.
> MCUSR:0x06; WDTCSR:0x69;
        * External Reset occurs
        * Brown-out Reset occurs
> Event: Type 0x06, TimeStamp = 0 msec (0 LOG); Store Index: 0x00;
> CLKSR:0xA2; CLKPR:0x00;
> 8MHz:  OSCCAL0:0x10; OSCTCAL0A:0x36; OSCTCAL0B:0x80;
> 32KHz: OSCCAL1:0x01;
> I2C_Device_EEPROM_Init; register to I2C device index 0;
> I2C_Device_ADC_Init; register to I2C device index 1;
> I2C_Device_GPI_Init; register to I2C device index 2;
> I2C_Device_SRAM_Init; register to I2C device index 3;
> I2C slave module Init. Slave base address: 0x70; Emulate x4 I2C devices.
> BMC reset detected.
> Event: Type 0x20, TimeStamp = 7270 msec (6 LOG); Store Index: 0x01;
> Asserted BMC CORST# .
> Turn-off flash power.
> Wait 5 msec ...
> Turn-on flash power.
> Wait 5 msec  ...
> Release CORST#.
> Wait for the second pulse on SPILOAD# ...
> Done.
```

Nuvoton provides comprehensive service and support.
For product information and technical assistance, contact the nearest Nuvoton center.

Headquarters
No. 4, Creation Rd. 3
Science-Based Industrial Park
Hsinchu, Taiwan, R.O.C
TEL: 886-3-5770066
FAX: 886-3-5665577
http://www.nuvoton.com.tw (Chinese)
http://www.nuvoton.com English)

Nuvoton Technology Corporation America
2727 North First Street
San Jose, CA 95134, U.S.A.
TEL: 1-408-544-1718
FAX: 1-408-544-1787

Nuvoton Technology (Shanghai) Ltd.
27F, 2299 Yan An W. Rd.
Shanghai, 200336 China
TEL: 86-21-62365999
FAX: 86-21-62365998

Taipei Office
1F, No.192, Jingye 1st Rd.
Zhongshan District
Taiwan City 104, Taiwan, R.O.C.
TEL: 886-2-2658-8066
FAX: 886-2-8751-3579

Winbond Electronics Corporation Japan
NO. 2 Ueno-Bldg., 7-18, 3-chome
Shinyokohama Kohoku-ku
Yokohama, 222-0033
TEL: 81-45-4781881
FAX: 81-45-4781800

Nuvoton Technology (H.K.) Ltd.
Unit 9-15, 22F, Millennium City 2,
378 Kwun Tong Rd.
Kowloon, Hong Kong
TEL: 852-27513100
FAX: 852-27552064

For Advanced PC Product Line information contact: APC.Support@nuvoton.com

www.nuvoton.com