Mixed Signal MCU
MD6603 Silicon Errata
Contents

Contents.................................................................................................................................................. 2

1. Restrictions on Flash Memory Controller.......................................................................................... 3
   1.1. Usage Notes on Erasing or Programming (Writing) Immediately after Re-protection...................... 3

2. Restrictions on 8051 CPU.................................................................................................................... 3
   2.1. Restrictions on XDATA BUS Buffer ............................................................................................... 3
   2.2. Conflict between 16-bit Register Write and Interrupt .................................................................... 4
   2.3. Access to RAM1 Area .................................................................................................................... 5
   2.4. Access to XDATA Space ................................................................................................................ 5
   2.5. MOVX Instructions to Peripheral Registers of XDATA Space ...................................................... 5
   2.6. EPU Operation While CPU Is Accessing UART Register ............................................................. 6
       2.6.1. EPU Access Failure by XDATA BUS Conflict ......................................................................... 6
       2.6.2. Usage Notes on Two Consecutive Accesses with Word Access Instruction ...................... 8

3. Restrictions on POC (PWM Output Controller)................................................................................. 9
   3.1. Operational Restrictions on Using Clear-wait Function ................................................................ 9

4. Restrictions on System Controller (SYSC)......................................................................................... 9
   4.1. Usage Notes on Returning from Sleep Mode ................................................................................ 9

5. Restrictions on 12-bit SAR ADC....................................................................................................... 11
   5.1. Restrictions on Sampling Cycle Settings ..................................................................................... 11

Important Notes....................................................................................................................................... 12

Revision History..................................................................................................................................... 13
This document describes all the silicon updates thus far to the functional specifications for the MD6603. The following usage notes and restrictions must be taken into account when designing your product.

1. Restrictions on Flash Memory Controller

1.1. Usage Notes on Erasing or Programming (Writing) Immediately after Re-protection

- Details
  Erasing or programming (writing) immediately after re-protection of the flash memory normally starts after a wait period. The duration of this wait period depends on the value* of the FMTIME register. Even after a wait period, erasing and programming (writing) to the flash memory can be successfully executed. If not re-executing the re-protection, no wait period occurs before subsequent erasing or programming (writing) is performed.

  * Specified by $FMTIME = \frac{CLKFAST}{1 \times 10^6} - 1$; if CLKFAST = 60 MHz and FMTIME = 59, the wait period is 32 ms.

- Workaround
  This malfunction can be avoided if not executing the re-protection.
  The LSI can perform erasing and programming (writing) to the flash memory without protection release when operated according to programs in the flash memory itself. Consequently, for the operations executed by the programs in the flash memory, the LSI does not require any protection release or re-protection on the flash memory.

- Tool-based Workaround
  None.

2. Restrictions on 8051 CPU

2.1. Restrictions on XDATA BUS Buffer

- Details
  While a program in a RAM on a bus is being executed, the functions of the XDATA BUS buffer cannot be used.

- Workaround
  None.

- Tool-based Workaround
  None.
2.2. Conflict between 16-bit Register Write and Interrupt

- Details

Writing to the 16-bit registers must not conflict with any interrupt acceptance; therefore, before writing to the higher bytes of the 16-bit registers (SFR BUS, XDATA BUS), set the INTMST.INTME bit to 0 to disable any interrupt acceptance. Then, insert two NOP instructions and write to the higher bytes. After writing to the higher bytes, set the INTMST.INTME bit to 1 to re-enable interrupt acceptance.

- Workaround

When writing data in C to a 16-bit width register which is assigned to the same address of the higher and lower bits in the SFR or the XDATA space (or both), implement the workarounds listed below.

1. In the first line of codes in the defines.h file, define the following macro (hereafter “Macro (1)” for short).

```c
#define WRITE16(RegisterName, Value)
do{
    const uint16_t value = (Value);
    __unsafe_##RegisterName = (uint8_t)value;
    if((INTMST & 1)){
        INTMST = INTMST & 0xfe;
        __asm__("nop");
        __asm__("nop");
        __unsafe_##RegisterName = (uint8_t)(value >> 8);
        INTMST = INTMST | 0x01;
    }else{
        __unsafe_##RegisterName = (uint8_t)(value >> 8);
    }
}while(0)
```

2. To prevent the 16-bit-width register from being written without Macro (1), specify const within the code that defines the registers in the defines.h file as below. By specifying the const as follows, a compile error occurs when the 16-bit width register is written without Macro (1).

```c
const __sfr __at (0xC4) DSP0_R0 ;
__sfr __at (0xC4) __unsafe_DSP0_R0 ;
...
const __xdata __at(0xF788) uint8_t DSP0R8;
__xdata __at(0xF788) uint8_t __unsafe_DSP0R8;
```

3. When writing data to the 16-bit width register, use Macro (1) as shown below.

```c
WRITE16(BUF_A0_LH, 0x1234); //SFR
WRITE16(DSP0_C0_HL, 0xabcd); //XDATA
```

- Tool-based Workaround

MD Studio provides the prewritten codes (in Workarounds (1) and (2) above) within the defines.h file, which is generated when you create a new project. However, note that the defines.h file is not supplied when you create a new project without templates.
2.3. Access to RAM1 Area

- **Details**
  During the EPU operation, accessing to the 512-byte RAM1 area (0x0400 to 0x05FF) from the CPU may result in conflicts, thus causing malfunctions.

- **Workaround**
  Do not access the 512-byte RAM1 area (0x0400 to 0x05FF) from the CPU during the EPU operation.

- **Tool-based Workaround**
  None.

2.4. Access to XDATA Space

- **Details**
  During the EPU operation, accessing to addresses of the XDATA space, ranging from 0xC000 to 0xDFFF, by the CPU may result in conflicts, thus causing malfunctions.

- **Workaround**
  Do not access the XDATA space addresses (0xC000 to 0xDFF) from the CPU during the EPU operation.

- **Tool-based Workaround**
  None.

2.5. MOVX Instructions to Peripheral Registers of XDATA Space

- **Details**
  During the EPU operation, executing two consecutive MOVX instructions by the CPU to a peripheral register of the XDATA space may result in conflicts, thus causing malfunctions. The examples below show code sequences using two consecutive MOVX instructions.

  **Example 1:** Write the same data consecutively to the same address of the XDATA peripheral register from the CPU.
  ```
  movx @dptr, a (Write)
  movx @dptr, a (Write)
  ```

  **Example 2:** Writing to/reading from the same address of the XDATA peripheral register consecutively by the CPU.
  ```
  movx @dptr, a (Write)
  movx a, @dptr (Read)
  ```

- **Workaround**
  Inserting a different instruction other than MOVX (e.g., NOP) between the two MOVX instructions can avoid malfunctions from such code sequences.

  When the program is written in C, implement the workaround as follows. When writing two consecutive MOVX instructions by the CPU to a peripheral register of the XDATA space during the EPU operation, insert a NOP instruction, `"\asm\"("nop")\";\"`, between the consecutive accesses by the CPU.

  **Example 1:** Write the same data consecutively to the same address of the XDATA peripheral register from the CPU.
  ```
  movx @dptr, a (Write)
  nop \leftarrow Insert a NOP instruction
  movx @dptr, a (Write)
  ```
Example 2: Writing to/reading from the same address of the XDATA peripheral register consecutively by the CPU.

```assembly
movx @dptr, a  ; (Write)
nop
movx a, @dptr  ; (Read)
```

- **Tool-based Workaround**
  When you use MD Studio to implement the workaround listed below, the user-programmable workarounds described above are unnecessary.

1. In SDCC, the code “-peep-file md6603.peep” is added as a compile option.
2. The md6603.peep file is automatically created from a skeleton code and placed under the hierarchy of the source file. Make sure that the md6603.peep file is placed under the hierarchy of the source file. The md6603.peep file contains the codes as given below.

```assembly
replace {
  movx @dptr,a
  movx @dptr,a
} by {
  movx @dptr,a
  nop
  movx @dptr,a
}
replace {
  movx @dptr,a
  movx a,@dptr
} by {
  movx @dptr,a
  nop
  movx a,@dptr
}
```

2.6. **EPU Operation While CPU Is Accessing UART Register**

2.6.1. **EPU Access Failure by XDATA BUS Conflict**

- **Details**
  If the EPU attempts to access the XDATA space while the CPU is accessing the UART register, the buses will have conflicts thus the EPU access fails.

- **Workaround**
  1. Access the UART by the EPU (alternative access for the CPU; see Table 2-1)
     This workaround is to permit the EPU to access the UART registers instead of the CPU. Handshaking with the CPU can control one thread of the EPU, thus permitting an alternative access to the UART registers by the EPU. Flags for handshaking, data addresses to be accessed are then read/written by the CPU and EPU from/to the following: SPR on the SFR BUS; the RAM0 (0x0000 to 0x03FF).
  2. Access the UART by the CPU (see Table 2-1)
     This workaround is to perform two consecutive accesses to the XDATA space by the EPU, thus avoiding access fails due to bus conflicts. The first access succeeds or fails, but the second access succeeds. In this description, “fail” means that no access by the EPU is issued to the XDATA space. If an access fails, no writing will be executed when a write access was attempted, whereas undefined values will be read when a read access was attempted.
     When writing the instructions which activate the hardware (e.g., ADC trigger, EPU activation, UART transmission and reception, DSAC trigger, PWM re-trigger, etc.) to the peripheral function registers, write an ineffective value at the first time or implement Workaround (1) above.
(2)-1 In the case where thread switching occurs while an EPU thread is being processed:

- Perform byte access to the XDATA space by the EPU
  Read/write bytes by using the word access instruction W_SA mode (i.e., two consecutive accesses to the same address). For the usage notes on the word access instruction, see Section 2.6.2 below.

- Perform word access to the XDATA space by the EPU
  Accessing the 16-bit-width peripheral function registers (i.e. consecutive accessing to the lower and higher bytes of the registers) is prohibited. When accessing the 16-bit-width peripheral function registers, implement Workaround (1) above.

(2)-2 In the case where no thread switching occurs while an EPU thread is being processed:

- When either of the following two conditions is met, perform two consecutive accesses to the XDATA space regardless of the byte or word access instruction: the condition in which only one thread is processed at once; or the condition in which no round-robin thread switching occurs while multiple threads are processed.

Table 2-1. List of Workarounds

<table>
<thead>
<tr>
<th>Workaround</th>
<th>Instructions to Be Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Access by EPU to UART (Alternative Access)</td>
<td>Byte Access Instruction by EPU</td>
</tr>
<tr>
<td>(2)-1 Access by CPU to UART (Thread Switching Occurs)</td>
<td>Substitute the word access instruction (see Section 2.6.2)</td>
</tr>
<tr>
<td>(2)-2 Access by CPU to UART (No Thread Switching Occurs)</td>
<td>Execute two consecutive byte instructions</td>
</tr>
</tbody>
</table>

• Tool-based Workaround

MD Studio supports the restriction on the EPU operation mentioned in this section (but offers only Workaround (1) above). MD Studio provides Workaround (1) through a program automatically created from a skeleton code to permit the EPU to access the UART register.

When accessing the UART register from the CPU, be sure to use the following functions to avoid the UART register being conflict-prone. The workaround program that permits the EPU to access the UART register consumes 44 bytes of the RAM1.

Here are detailed descriptions how to use the EPU:

- This workaround program requires the EPU Thread5. Round-robin settings on the Thread5 and other threads are user-selectable. For round-robin settings, set a proper group for thread priority control such that the Thread5 is executable. In addition, do not write to modify the Thread5 such as for adding an instruction. Note that the Thread5 is also used for the workaround program described in Section 4.1.

- Through the use of the RAM0 (0x0000 to 0x03FF), handshaking takes place to transmit address values and data between the CPU.

- The EPU Thread5 that allows the EPU to access the UART register is normally stopped. The Thread5 operates only when the CPU calls the functions, then stops again. (This means that it stops an instruction fetch of the EPU and lowers power consumption.)

- Write an address of the UART register as an argument, addr, used in the functions provided by MD Studio. When defining a register address, use the "ADDR_UART register name" code defined in the errata.h file.

The functions to be provided are as follows:

```c
//Read a value from the UART register and return the value.
uint8_t UART_Reg_RD(uint16_t addr);
//Write data to the UART register.
void UART_Reg_WR(uint16_t addr, uint8_t data);
```
- Including the errata files:

To implement workaround programs appropriately, select a setting from the following three combinations indicating which program must be enabled or disabled. The workaround programs include the UART access workaround and the sleep mode workaround (see Section 4.1). However, do not select a setting not listed below (i.e., the combination of “UART access workaround disabled; sleep mode workaround enabled”). If the improper combination is selected, the UART access workaround attempts to use the functions even during sleep mode operation, thus causing a build error.

Enable-or-disable combinations of workaround programs:

- UART access workaround enabled; sleep mode workaround enabled
  - Code used in the epu.easm file:

```asm
.include "errata_uart.easm"
.include "errata_sleep.easm"
```

- UART access workaround enabled; sleep mode workaround disabled
  (When a new project is created, the default epu.easm file contains the workaround code below.)
  - Code used in the epu.easm file:

```asm
.include "errata_uart.easm"
#.include "errata_sleep.easm"
```

- UART access workaround disabled; sleep mode workaround disabled
  - Code used in the epu.easm file:

```asm
#thread(5){
#    #Please set proper priority of thread 5.
#    priority:None
#    launch:None
#    pc:ERRATA_TH5_START
#    EIC:0
#}
#.include "errata_uart.easm"
#.include "errata_sleep.easm"
```

2.6.2. Usage Notes on Two Consecutive Accesses with Word Access Instruction

When you perform two consecutive accesses to the same address by using the word access instruction, each access will end up with two different results according to which of the accesses, read (LOADX) or write (STOREX), has been attempted.

**Read Access:** LOADX.W_SA Rn, @AddrX

- The first and second accesses succeed
- The first access fails (a LOAD value is undefined); the second access succeeds

**Write Access:** STOREX.W_SA @AddrX, Rn

- The first and second accesses succeed
- The first access fails (access is lost); the second access succeeds
If both the first and second accesses succeeded, the first write access may cause malfunctions. Therefore, the following workarounds must be implemented, for proper writing to the registers by the EPU.

- Writing to the setting register of the peripheral function:
  Write the same values at the first and second accesses.
- Clearing the flags of the peripheral function registers:
  Write 0 at the first time, then clear at the second time.
- Writing the instructions which activate the hardware (e.g., ADC trigger, EPU activation, UART transmission and reception, DSAC trigger, PWM re-trigger, etc.) to the peripheral function registers:
  Write an ineffective value, or implement Workaround (1) in Section 2.6.1.

3. Restrictions on POC (PWM Output Controller)

3.1. Operational Restrictions on Using Clear-wait Function

- Details
  The following restrictions are implemented when clearing the POC control in the case where the control delay addition function is used and the clear-wait function is enabled (i.e., setting the POCDDTCn register to either of 0x81, 0x82, or 0x83):
  (1) Auto-release by PWM is not available. Be sure to set POCCRn.ARELEN = 0.
  (2) Before clearing the POC control by writing 1 to the POCSTS.PWMnF bit, read the POCSTS register to make sure that the bit to be cleared is set to 1 (or, currently being controlled). Otherwise, the LSI may result in unexpected behaviors thus malfunctions.

- Workaround
  None.

- Tool-based Workaround
  None.

4. Restrictions on System Controller (SYSC)

4.1. Usage Notes on Returning from Sleep Mode

- Details
  When the LSI falls into the operation states listed below, the CPU does not return from the sleep mode.
  (1) The condition in which the CPU receives a high-priority interrupt signal before or after writing an instruction enabling the sleep mode to a register.
  (2) The condition in which the CPU writes an instruction enabling the sleep mode to a register while a high-priority interrupt is being processed.

  Under the operation states listed below, the CPU returns from the sleep mode only when it receives a high-priority interrupt. However, the CPU does not return from the sleep mode when it receives a low-priority interrupt.
  (1) The condition in which the CPU receives a low-priority interrupt signal before or after writing an instruction enabling the sleep mode to a register.
  (2) The condition in which the CPU writes an instruction enabling the sleep mode to a register while a low-priority interrupt is being processed.
In a normal operation state (i.e., no interrupt is being processed), when the CPU receives no interrupt signal before or after writing a sleep-mode-enabling instruction to a register, it returns from the sleep mode by an interrupt signal regardless of the signal’s priority level.

A high-priority interrupt is an interrupt whose bit corresponding to the INTLVLn register of the INTC is set to 1. A low-priority interrupt is an interrupt whose bit corresponding to the INTLVLn register of the INTC is set to 0.

- **Workaround**
  
  The following workarounds should be implemented when the sleep mode is used.

  1. **Workaround before CPU entering into sleep mode**
     
     Before the CPU enters the sleep mode, disable interrupts whose generation timing is non-CPU-controllable, such as those generated by external signals input to the comparators and GPIO. When using interrupts whose generation timing is CPU-controllable, set them not to be generated during the execution of a sleep-mode-enabling instruction.

  2. **Workaround for sources used when CPU returning from sleep mode**
     
     Use interrupts whose generation timing is CPU-controllable as sources for returning from the sleep mode. When using interrupts whose timing is uncontrollable as recovery sources, enable interrupts by the EPU after the CPU enters the sleep mode. The EPU cannot access the registers of the INTC; therefore, the interrupt functions of the peripheral modules which issue interrupts should be enabled or disabled individually. It is recommended to define an interrupt used for a recovery source from the sleep mode as a high-priority interrupt.

- **Tool-based Workaround**
  
  MD Studio provides the following functions as skeleton codes: “void Sleep_Init(void)” as the initialization function which executes the Goto_Sleep function; “void Goto_Sleep(void)” as the sleep-mode-enabling function. When using these functions, include the following two files into the the epu.easm file (i.e., an EPU assembly file): the errata_uart.easm and errata_sleep.easm files.

  When the errata_sleep.easm file is output from the skeleton code, it contains the include statement coded as follows: “#include errata_sleep.easm”. When using the sleep mode, delete the "#" symbol in the include statement mentioned above to make its code description executable. When the errata_sleep.easm file is included into the epu.easm file, the LSI automatically executes the Sleep_Init() within the main function. At this time, this workaround program consumes 74 bytes of the RAM0 area. By using the user-written code that calls the Goto_Sleep() function, the LSI can properly enter and return from the sleep mode by an interrupt.

  The Goto_Sleep() function runs as follows:

  1. Obtains the status data of all the enabled interrupts from peripheral modules, then stores it into the RAM0.
  2. Disables all of the interrupts from the peripheral modules.
  3. Activates the EPU Thread5, which is also used in the workaround program in Section 2.6.1.
  4. The CPU enters the sleep mode.
  5. The Thread5 re-enables the interrupts whose status values were enabled at the time when stored in the RAM0. Then, the Thread5 enters a wait state.
  6. Returns to the main function after waking from the sleep mode.

  When the errata_sleep.easm file is included, the sleep mode workaround uses 100 bytes of the RAM0. The EPU Thread5 is used as a dedicated thread for the workaround programs provided by MD Studio. Do not change the settings other than thread priorities.

  When the WDT module is used as an interval timer, the Goto_Sleep() function stops the counting operation of the WDT to disable interrupts. Therefore, when the Goto_Sleep() function is executed, the count period of the WDT’s interval timer becomes longer than its original setting.

  However, do not execute the Goto_Sleep() function within the interrupt sequence. Otherwise, the LSI may not be able to return from the sleep mode. When implementing the sleep mode workaround, be sure to use the functions defined in Section 2.6.1, including “UART_Reg_RD(uint16_t addr)” and “UART_Reg_WR(uint16_t addr, uint8_t data)”, to access the UART register.
5. Restrictions on 12-bit SAR ADC

5.1. Restrictions on Sampling Cycle Settings

- **Details**
  The sampling cycles listed below cannot provide proper converted values when used in the SHTIME bit, which specifies a sampling cycle of the ADNSMPmn register.
  5, 37, 69, 101, 133, 165, 197, 229

- **Workaround**
  The sampling cycles listed below are prohibited from using in the SHTIME bit, which specifies a sampling cycle of the ADNSMPmn register.
  5, 37, 69, 101, 133, 165, 197, 229

- **Tool-based Workaround**
  None.
Important Notes

- All data, illustrations, graphs, tables and any other information included in this document (the “Information”) as to Sanken’s products listed herein (the “Sanken Products”) are current as of the date this document is issued. The Information is subject to any change without notice due to improvement of the Sanken Products, etc. Please make sure to confirm with a Sanken sales representative that the contents set forth in this document reflect the latest revisions before use.
- The Sanken Products are intended for use as components of general purpose electronic equipment or apparatus (such as home appliances, office equipment, telecommunication equipment, measuring equipment, etc.). Prior to use of the Sanken Products, please put your signature, or affix your name and seal, on the specification documents of the Sanken Products and return them to Sanken. When considering use of the Sanken Products for any applications that require higher reliability (such as transportation equipment and its control systems, traffic signal control systems or equipment, disaster/crime alarm systems, various safety devices, etc.), you must contact a Sanken sales representative to discuss the suitability of such use and put your signature, or affix your name and seal, on the specification documents of the Sanken Products and return them to Sanken, prior to the use of the Sanken Products. The Sanken Products are not intended for use in any applications that require extremely high reliability such as: aerospace equipment; nuclear power control systems; and medical equipment or systems, whose failure or malfunction may result in death or serious injury to people, i.e., medical devices in Class III or a higher class as defined by relevant laws of Japan (collectively, the “Specific Applications”). Sanken assumes no liability or responsibility whatsoever for any and all damages and losses that may be suffered by you, users or any third party, resulting from the use of the Sanken Products in the Specific Applications or in manner not in compliance with the instructions set forth herein.
- In the event of using the Sanken Products by either (i) combining other products or materials or both therewith or (ii) physically, chemically or otherwise processing or treating or both the same, you must duly consider all possible risks that may result from all such uses to advance and proceed therewith at your own responsibility.
- Although Sanken is making efforts to enhance the quality and reliability of its products, it is impossible to completely avoid the occurrence of any failure or defect or both in semiconductor products at a certain rate. You must take, at your own responsibility, preventative measures including using a sufficient safety design and confirming safety of any equipment or systems in/for which the Sanken Products are used, upon due consideration of a failure occurrence rate and derating, etc., in order not to cause any human injury or death, fire accident or social harm which may result from any failure or malfunction of the Sanken Products. Please refer to the relevant specification documents and Sanken’s official website in relation to derating.
- No anti-radioactive ray design has been adopted for the Sanken Products.
- The circuit constant, operation examples, circuit examples, pattern layout examples, design examples, recommended examples, all information and evaluation results based thereon, etc., described in this document are presented for the sole purpose of reference of use of the Sanken Products.
- Sanken assumes no responsibility whatsoever for any and all damages and losses that may be suffered by you, users or any third party, or any possible infringement of any and all property rights including intellectual property rights and any other rights of you, users or any third party, resulting from the Information.
- No information in this document can be transcribed or copied or both without Sanken’s prior written consent.
- Regarding the Information, no license, express, implied or otherwise, is granted hereby under any intellectual property rights and any other rights of Sanken.
- Unless otherwise agreed in writing between Sanken and you, Sanken makes no warranty of any kind, whether express or implied, including, without limitation, any warranty (i) as to the quality or performance of the Sanken Products (such as implied warranty of merchantability, and implied warranty of fitness for a particular purpose or special environment), (ii) that any Sanken Product is delivered free of claims of third parties by way of infringement or the like, (iii) that may arise from course of performance, course of dealing or usage of trade, and (iv) as to the Information (including its accuracy, usefulness, and reliability).
- In the event of using the Sanken Products, you must use the same after carefully examining all applicable environmental laws and regulations that regulate the inclusion or use or both of any particular controlled substances, including, but not limited to, the EU RoHS Directive, so as to be in strict compliance with such applicable laws and regulations.
- You must not use the Sanken Products or the Information for the purpose of any military applications or use, including but not limited to the development of weapons of mass destruction. In the event of exporting the Sanken Products or the Information, or providing them for non-residents, you must comply with all applicable export control laws and regulations in each country including the U.S. Export Administration Regulations (EAR) and the Foreign Exchange and Foreign Trade Act of Japan, and follow the procedures required by such applicable laws and regulations.
- Sanken assumes no responsibility for any troubles, which may occur during the transportation of the Sanken Products including the falling thereof, out of Sanken’s distribution network.
- Although Sanken has prepared this document with its due care to pursue the accuracy thereof, Sanken does not warrant that it is error free and Sanken assumes no liability whatsoever for any and all damages and losses which may be suffered by you resulting from any possible errors or omissions in connection with the Information.
- Please refer to our official website in relation to general instructions and directions for using the Sanken Products, and refer to the relevant specification documents in relation to particular precautions when using the Sanken Products.
- All rights and title in and to any specific trademark or tradename belong to Sanken and such original right holder(s).
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date of Issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Apr. 6, 2018</td>
<td>Initial release</td>
</tr>
</tbody>
</table>

Revision History