Application Program	**Bus Interface Modules M130, M131, M132 and M135**
Runtime Environment for	**KNX-Processors 184/01, 184/11 and 184/21**

Content of this document:

Document-Version: 2.1

1 Memory map

To provide compatibility with ETS the operating system does a memory translation from bus addresses to the internal addresses of the microcontroller. That means that for example the start of the address table for the ETS is 0x0116 and in reality it is 0x8116 in the microcontroller.
The following memory map tables show respectively the real memory addresses in the microcontroller, the addresses which are used on the bus to access them, the type and for what they are used.

1.1 BIM M 130, BIM M 135 and Chipset 184/01

Address in F0534	Bus access address	Used by	Type
0xFEDF 0xFBD0	**n.a.**	**System**	**RAM**
0xFBCF 0xFB00	**n.a.**	**User (Stack and Variables)**	**RAM**
0xF7FF 0xF400	**n.a.**	**System**	**RAM**
0xBFFF 0xA000	**n.a.**	**System (Reserved)**	**Flash**
0x9FFF 0x8116	**0x1FFF 0x0116**	**User (Appl.-Code)**	**Flash**
0x8115 0x8000	**0xE915 0xE800**	**User (Appl.-InfoBlock)**	**Flash**
0x7FFF 0x7800	**0xE7FF 0xE000**	**User (RootCode)**	**Flash**
0x77FF 0x7700	**n.a.**	**System (JumpTable)**	**Flash**
0x76FF 0x0000	**n.a.**	**System**	**Flash**

1.2 BIM M 131 and Chipset 184/11

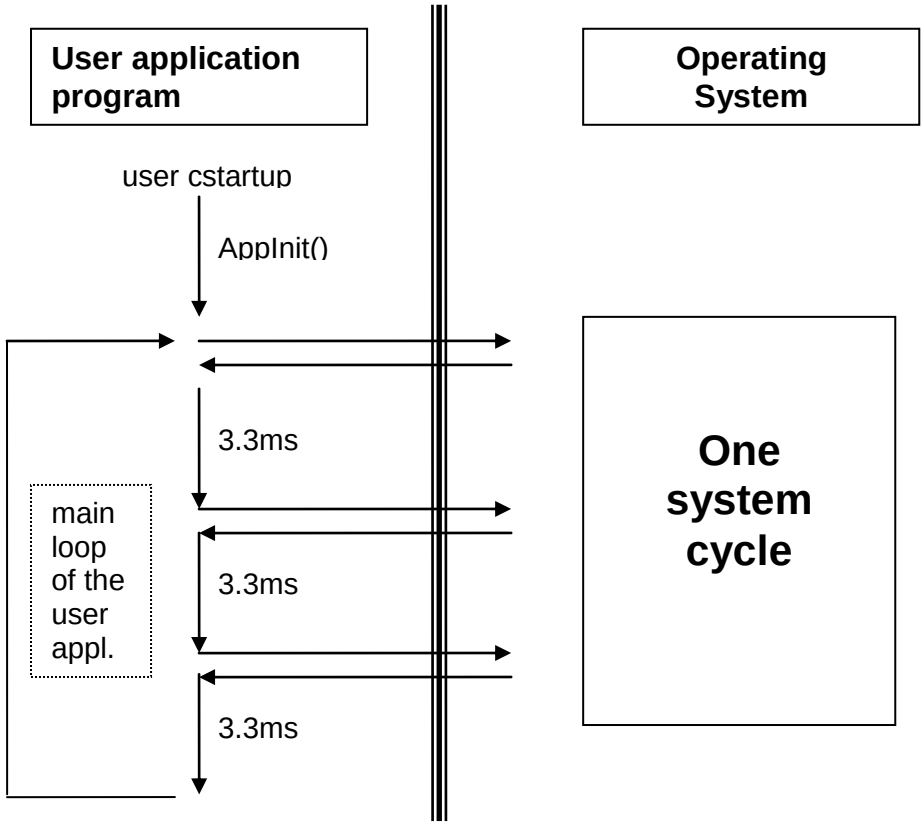
Address in F0535	Bus access address	Used by	Type
0xFEDF 0xFBD0	**n.a.**	**System**	**RAM**
0xFBCF 0xFB00	**n.a.**	**User (Stack and Variables)**	**RAM**
0xF7FF 0xF400	**n.a.**	**System**	**RAM**
0xF3FF 0xF000	**n.a.**	**User (Variables)**	**RAM**
0xEFFF 0xD000	**n.a.**	**System (Reserved)**	**Flash**
0xCFFF 0xC000	**n.a.**	**User (Appl.-Code)**	**Flash**
0xBFFF 0x8116	**0x3FFF 0x0116**	**User (Appl.-Code)**	**Flash**
0x8115 0x8000	**0xE915 0xE800**	**User (Appl.-InfoBlock)**	**Flash**
0x7FFF 0x7800	**0xE7FF 0xE000**	**User (RootCode)**	**Flash**
0x77FF 0x7700	**n.a.**	**System (JumpTable)**	**Flash**
0x76FF 0x0000	**n.a.**	**System**	**Flash**

1.3 BIM M 132 and Chipset 184/21

Address in F0537	Bus access address	Used by	Type
0xFEDF 0xFBD0	**n.a.**	**System**	**RAM**
0xFBCF 0xFB00	**n.a.**	**User (Stack and Variables)**	**RAM**
0xF7FF 0xF400	**n.a.**	**System**	**RAM**
0xF3FF 0xE000	**n.a.**	**User (Variables)**	**RAM**
0xBFFF 0x8000 (Bank 5)	**n.a.**	**System (Reserved)**	**Flash**
0xBFFF 0x8000 (Bank 4)	**n.a.**	**User (Appl.-Code)**	**Flash**
0xBFFF 0x8000 (Bank 3)	**n.a.**	**User (Appl.-Code)**	**Flash**
0xBFFF 0x8000 (Bank 2)	**0xBFFF 0x8000**	**User (Appl.-Code)**	**Flash**
0xBFFF 0x8000 (Bank 1)	**0x7FFF 0x4000**	**User (Appl.-Code)**	**Flash**
0xBFFF 0x8116 (Bank 0)	**0x3FFF 0x0116**	**User (Appl.-Code)**	**Flash**
0x8115 0x8000 (Bank 0)	**0xE915 0xE800**	**User (Appl.-InfoBlock)**	**Flash**
0x7FFF 0x7800	**0xE7FF 0xE000**	**User (RootCode)**	**Flash**
0x77FF 0x7700	**n.a.**	**System (JumpTable)**	**Flash**
0x76FF 0x0000	**n.a.**	**System**	**Flash**

2 Task switch system

To allow the implementation of the user application program in an endless loop the operating system provides a simple task switch system, that interrupts the user application program every 3.3ms for one complete system cycle.



The ‘TESTPIN2’ on the evaluation board is logical one while the user application is running and logical zero after it was interrupted. The macro ‘FORCE_TASKSWITCH’ can be used to force a task switch immediately, the macro ‘DIS_TASKSWITCH’ to disable the task switch and ‘EN_TASKSWITCH’ to enable it again. The task switch should only be disabled in critical sections, for example on access to communication objects that are larger than one byte. Normally at the beginning of a task switch interrupt all processor registers are saved on the stack and restored before the ‘RETI’ command is executed. This is not done here, because the Renesas 78K0 microcontroller has 4 times the complete set of processor registers (AX, BC, DE and HL). The actual register set is stored in the processor status word that is pushed on the stack of each task.

The switch between register sets is used to enhance performance. The four register sets are used as follows:

- RB0: for user task
- RB1: for system task
- RB2: in all interrupt functions
- RB3: for flash firmware

The user application program must not modify the register select flags in the processor status word!

3 User application program

The user application program has to be developed as a regular program for a Renesas 78K0 microcontroller, but with some modifications. To give the possibility to debug the user application program on the evaluation board there are some special differences to a normal program for the 78K0. First of all the linker file has no segment for the interrupt vectors because the user application program can not handle any interrupt directly. In the linker file the following segments are defined:

```
/*********************************************************************\
|*   D A T A                                                         *|
\*********************************************************************/
-Z(DATA)CSTACK+_CSTACK_SIZE#FBCF
-Z(DATA)NEAR_I,NEAR_Z,NEAR_N=FB00-FBCF
-Z(DATA)EXPRAM=...

/*********************************************************************\
|*   C O D E                                                         *|
\*********************************************************************/
-Z(CODE)PARAM=...
-Z(CODE)NEAR_ID,CONST,RCODE,CODE=...

-Z(CODE)COMTAB=...
-Z(CODE)ASSOCTAB=...
-Z(CODE)ADDRTAB=8116-...

-Z(CODE)APPINFOBLOCK=8000-8115
-Z(CODE)ROOTCODE=7800-7FFF
/*********************************************************************/
-Z(CODE)BCU2_JMP=7700-77FF
```

-Z(DATA) is for variables in ram and -Z(CODE) is for all data that are stored in flash.
The stack of the user application program starts always at 0xFBCF and grows down from there. ‘NEAR_I’ contains initialized, ‘NEAR_Z’ zero initialized and ‘NEAR_N’ non initialized global variables. The ‘EXPRAM’ specifies the expansion ram of the 78K0 microcontroller and is only available for the user application program in the BIM M 131 and M 132 as well as in chipset 184/11 and 184/21.
In ‘BCU2_JMP’ the jump table with the vector addresses for the API functions is located. The data in this segment is provided by the operating system. ‘ROOTCODE’ could be used for the BIM M 132. This bus interface module has banked flash. A switch between these flash banks has to be done in the root area. In the segment ‘APPINFOBLOCK’ the application info block is located which is described below. At ‘ADDRTAB’ the code for the address table, at ‘ASSOCTAB’ the code for the association table and at ‘COMTAB’ the code for the communication object table starts. These segments can be accessed from the bus via read and write requests starting from 0x0116 and the specified addresses can be changed according to the circumstances of the user application program but considering the restrictions of the memory map of the different bus interface modules.

The ‘NEAR_ID’ contains the value for the ram variables that have to be initialized at startup of the user application program through ‘cstartup’.
Const data is placed in ‘CONST’.
The segment ‘RCODE’ contains the ‘cstartup’ code and the runtime library code.
‘CODE’ contains the normal program code.
The user application programmer can change or divide the segments ‘ADDRTAB’, ‘ASSOCTAB’, ‘COMTAB’, ‘NEAR_ID’, ‘CONST’, ‘RCODE’ and ‘CODE’ if there is the need to do this, for example if the segment ‘PARAM’ is not used or must be increased. This segment is normally used to store parameters of the application program at fix addresses.

When the debug session is started in the IAR Embedded Workbench, the debugger tries to rewrite the reset vector so that it points to the entry point of the 'cstartup.asm' file. To prevent this the 'cstartup.asm' was modified in that way that if the project configuration 'DEBUG' is chosen a manual generated complete interrupt vector table is linked to the address where the normal interrupt vector table has to be. This is done with the 'ORG' directive in assembler. The interrupt vector table that is defined in the cstartup file in the user application program is exactly the table as it is in the BIM / chipset operating system. That means if the user application programmer starts the debug session the controller will start with the operating system which is already loaded in the microcontroller and not with the user application program. The operating system in turn will start the user application program. This is done by initializing the task switch system and an immediate task switch to the user application program. After this task switch the user application program starts at the entry point in the cstartup where the task switch will be disabled. This is done to allow the user application program to do all initializations that are necessary before the operating system does the next system cycle. In the cstartup of the user application program all ram variables that are located in the high speed ram are initialized either with zero or with their initial value. After this the main function is called where more initializations could be done, for example some init functions of the used API functionality. When the user application program has finished the initializations the macro ‘FORCE_TASKSWITCH’ has to be called to enable the task switch again and to do a system cycle immediately. The complete startup of the user application program from the beginning of the cstartup to the call of 'FORCE_TASKSWITCH' should not take more then 10ms.

Each user application program must have a user application information block. This block has always to be at 0x8000. In this block the user gives some information to the operating system which is described in the following:

```
typedef struct
{
    USHORT                  AIBVersion;
    BYTE ApplFirmwareVersion;
    BYTE ApplFirmwareSubVersion;
    void (*AppMain)         (void);
    void (*AppSave)         (void);
    void (*AppUnload)       (void);
    const CObjPtr*          pCObjects;
    BYTE*                   pRAMFlags;
    TIMER_TAB*              pUserTimerTab;
    const INTERFACE_ROOT*   pUsrIntObjRoot;
    ParamMgmt*              pUsrParamMgmt;
    USHORT                  WatchDogTime;
} AppInfoBlock;
```

The ‘AIBVersion’ is the version of the application info block. At the moment 0x0001 has to be specified. ‘ApplFirmwareVersion’ and ‘ApplFirmwareSubVersion’ could be read from bus via property 3/204 and 3/205. The count scheme is completely free to the user application programmer. ‘AppMain’ is the entry point of the user application program, ‘AppSave’ the pointer to the save routine and ‘AppUnload’ the pointer to the unload routine. ‘AppSave’ is called on loss of power and ‘AppUnload’ is called if an unload event occurs, for example if a download is done by ETS software. ‘CObjPtr’ is the pointer to the communication object pointer table. ‘pRAMFlags’ is a pointer to the ram flags. ‘pUserTimerTab’ is a pointer to the user timer table. If no user timer table is used set this value to ‘NULL’. ‘pUsrIntObjRoot’ is a pointer to the root table of the user interface objects (properties). If no user properties are used set this value to ‘NULL’. ‘pUsrParamMgmt’ is a pointer to the parameter management for the user application program. If this feature is not used the value has to be ‘NULL’. The user application program can specify a ‘WatchDogTime’. If this value is not 0x0000 ‘U._TriggerWatchDog()’ must be called within the specified time. If this is not done the operating system resets the device.

4 Communication object handling

All communication objects of the application program are defined in table ‘EE_CommsTab’. The table starts with a byte that specifies the number of communication objects. The next byte is only implemented for compatibility and should be 0x00. After this per communication object three bytes are defined. For the first byte ‘VALID_BCU2ADR’ is always used. This is done for compatibility. The second byte specifies the communication object flags and the third byte the size of the communication object. See ‘ConfigByte’ and ‘Typebyte’ in ‘BIM_M13x.h’ for the possible values.
The interface between the user application program and the operating system are the ram flags. The user application program must provide one nibble of ram for each communication object. This is done by defining the following array:

```
BYTE RAMFlags[(NUM_OF_COM_OBJ / 2) + 1];
```

That means an array is created that has half byte of the number of communication objects. The pointer to this array must be specified in the application info block. The API functions that are described in the API-Reference at the point “Object-Handling” act on the ram flags. Each time an update for a communication object was received the operating system sets the ram flags and stores the new value in the user application ram. Therefore the user application program has to specify a table with pointers for each communication object (name: CObjects). The first pointer must point to a byte that contains the number of entries in the table. The number of entries must be the same as in ‘EE_CommsTab’. The next pointer is the pointer to a ram variable for communication object 0, the next pointer for communication object 1 and so on.
The address of this table has to be specified in the application info block.

5 How to make a RELEASE

There are two possibilities that could be done with a ready developed BIM M 13x application. First a S19 file could be generated for an ETS database entry. For this the KNX manufacturer tool is necessary. In the manufacturer tool the S19 file could be imported for a new database entry that in turn could be imported in an ETS project after the registration of the database entry.
The other method is to flash the BIM M 13x device with a flash programmer (e.g. Renesas PG-FP5).
The procedure for these both possibilities is explained in the following.

5.1 S19 for download via bus

To generate a S19 file that could be imported in the KNX manufacturer tool it is necessary to convert the memory addresses and to add ‘load controls’. The memory conversion has to be done because as mentioned above the BIM / chipset operating system does a memory translation for ETS compatibility. Therefore a byte located at 0x8116 in the microcontroller has to be set at 0x0116 in the S19 file for bus download because the operating system will set it back to 0x8116.
The ‘load controls’ are necessary to control the processing of the address, association and communication object description table and the application program.
The memory conversion and adding of ‘load controls’ is done by a command line tool installed with the BIM-Tools. A project generated with the BIM-Tools Project Wizard has already set the “Post-build command line”-option to the command line tool aioc.exe with the necessary command line arguments. It is specified via “Project”, “Options…”, dialog “Build Actions” in the text box “Post-build command line” as follows:

<program path>\aioc.exe -t modifier --targs <comment> <config file> <input file> <output file>

- <program path>: means the complete path to the “aioc.exe”
- <comment>: a string that will be inserted as S19 comment
- <config file>: xml configuration file for the modifier (normally "$PROJ_DIR$\config.xml")
- <input file>: the input S19 file (generated from IAR Embedded Workbench)
- <output file>: the output file name

The xml configuration file has the following structure:

```
<?xml version="1.0" encoding="utf-8" ?>
<BIM_M13x_S19_Modification
xmlns="http://tempuri.org/BIM_M13x_S19_Modification.xsd">
    <MemoryConversion>
      ...
    </MemoryConversion>
    <LoadControls>
      ...
    </LoadControls>
</BIM_M13x_S19_Modification>
```

The ‘MemoryConversion’ element has one or more entries of the element ‘MemoryConvEntry’ that is defined as follows:

```
<MemoryConvEntry>
      <Start>...</Start>
      <End>...</End>
      <Offset>...</Offset>
</MemoryConvEntry>
```

The ‘Start’ element defines the start address and the ‘End’ element the end address of the memory range that should be converted. The conversion is done by adding the value of the ‘offset’ element to the addresses within the specified memory range.

The ‘LoadControls’ element has different load control entries which are defined in the following. The load controls are read out from ETS and control how the download of the application program will be done:

- `<Connect />`
  Generates a connect to the device

- `<Disconnect />`
  Generates a disconnect to the device

Siemens AG
Infrastructure and Cities Sector,
Building Technologies
Control Products and Systems
P. O. Box 10 09 53,
D-93009 Regensburg

- ```
  <PropertyCompare>
      <ObjectIndex>00</ObjectIndex>
      <PropertyID>4E</PropertyID>
      <StartIndex>001</StartIndex>
      <NumOfElements>1</NumOfElements>
      <Data>010203040506</Data>
  </PropertyCompare>
  ```
  Generates a ‘property compare’. In this example the value of the property 0/78 (Hex: 0x4E) is compared with the value ‘0x01 0x02 0x03 0x04 0x05 0x06’. The property 0/78 of the BIM is the ‘hardware type’ and is a ‘generic 06’ property with 6 bytes. It can be written when a new device with the BIM M 13x is manufactured. The property compare with this property can be used to ensure that the ETS will only download the right application program to it, because if the specified ‘data’ in the load control is not the same as in the device the ETS will stop the download with an error.

- ```
  <Unload>
      <StateMachine>...</StateMachine>
  </Unload>
  ```
  Sets the specified state machine to ‘unload’. That means the processing of the specified part will be stopped. Possible values are ‘Addr’ for the address table, ‘Assoc’ for the association table and ‘App’ for the communication object description table and the application program.

- ```
  <Load>
      <StateMachine>...</StateMachine>
  </Load>
  ```
  Sets the specified state machine to ‘loading’. After this the memory segment of the specified part can be filled with new data. Possible values are ‘Addr’ for the address table, ‘Assoc’ for the association table and ‘App’ for the communication object description table and the application program.

- ```
  <LoadComleted>
      <StateMachine>...</StateMachine>
  </LoadComleted>
  ```
  Sets the specified state machine to ‘Loaded’. That means the processing of the specified part will be started again. Possible values are ‘Addr’ for the address table, ‘Assoc’ for the association table and ‘App’ for the communication object description table and the application program.

- <DataSegment>
  <StateMachine>...</StateMachine>
  <StartAddr>...</StartAddr>
  <EndAddr>...</EndAddr>
  </DataSegment>
  This load control specifies an address range that will be downloaded to the device. The ETS only loads the data that are specified within this range. Data in the S19 file that are not in a load control data segment will not be downloaded. ETS will only load the data that is available in the S19 file within the specified segment. That means vacancies in a specified memory range are left out. Possible values for the element 'StateMachine' are 'Addr' for the address table, 'Assoc' for the association table and 'App' for the communication object description table and the application program. The elements 'StartAddr' and 'EndAddr' specify the start and end address of the segment. The values are the original addresses of the microcontroller. They will be converted according to the memory conversion entries.

- <TablePointer>
  <StateMachine>...</StateMachine>
  <StartAddr>...</StartAddr>
  </TablePointer>
  This load control tells the operating system the start address of the specified table. Possible values for the element 'StateMachine' are 'Addr' for the address table, 'Assoc' for the association table and 'App' for the communication object description table. The value of the element 'StartAddr' is the real address in the microcontroller. It will be converted according to the memory conversion entries. The ETS can change the data of the tables. Therefore the data of the three tables will be different to the data specified in the development environment.

- <AppData>
  <PEIType>FF</PEIType>
  <Manufacturer>...</Manufacturer>
  <AppID>...</AppID>
  <AppVersion>...</AppVersion>
  </AppData>
  This load control writes application specific data. The value of the element "PEIType" normally specifies the PEI-Type that is necessary to allow a start of the application. Because a BIM is a fix mounted device the operating system of the BIM does not check this value. The value of the elements "Manufacturer", "AppID" and "AppVersion" could later be read out from the device via property 3/13.

5.2 Download with flash programmer

To generate a S19 file that can be downloaded with the Renesas flash programmer the IAR Embedded Workbench must output a normal S19 (S37 in case of BIM M 132 / chipset 184/21) file. This file has not to be converted as it was done in the previous point because the flasher writes the data directly to the specified addresses without the operating system of the BIM. However after the download a few bus telegrams are necessary because the operating system does not know the start addresses of the address, association and communication object description table. Therefore the xml configuration file for the modifier has to be changed that it has no data segments but only the table pointers and the 'Unload', 'Load', 'LoadCompleted' and 'AppData' load controls.
The project wizard automatically generates beside the 'config.xml' file a 'config_pgfp4.xml' file, too. This file can be used regardless which flash programmer is used to program the device (switch to "Release" configuration in IAR EWB and replace the "config.xml" statement with "config_pgfp4.xml" in the "Post-build command line" in "Project" → "Options…" → "Build Actions").
In the following the configuration of the Renesas (former NEC) flash programmer PG-FP5 will be described.

5.2.1 PG-FP5

The software for the PG-FP5 flash programmer is used to download the created S19 file. First it is necessary to generate a setup file (*.esf). This is done via menu "Device" → "Setup…". On the tab "Target" click on button "New" in the area "Parameter file and Setting file". Enter a name for the *.esf file and select the right *.pr5 file according to the microcontroller that is used. The *.pr5 file has to be in the same directory where the *.esf file will be stored. To choose the right *.pr5 file see table below where the used microcontrollers are shown. Click on button "Save", select in the area "Object HEX file" the *.S19 (*.S37) file that was generated from the IAR Embedded Workbench and enable "Erase memory before download". Next go to tab "Standard" and configure the area "Communication interface to device" with Port "CSI-Internal-OSC" and Speed "2500kHz". In area "Operation Mode" select "Block" instead of "Chip" programming to avoid erasing the already installed operating system in the chip. The following table shows the start and end addresses of the different BIMs and chipsets.

BIM	Chipset	Microcontroller	Start address	End address
M130, M135	184/01	µPD78F0534 µPD78F0534A	0x7800	0x9FFF
M131	184/11	µPD78F0535 µPD78F0535A	0x7800	0xBFFF (0xCFFF)
M132	184/21	µPD78F0537 µPD78F0537A	0x7800	0x13FFF (0x1BFFF)

Enable the check box "Show Address" to see addresses instead of block numbers in the start and end drop down boxes. On the tab "Advanced" select "On Target" and "Vdd monitoring" in area "Supply voltage" and in area "Command options" enable "Blank check before Erase" and "Verify after Program" if you want to do this.

Last click on button "OK" to store the download file and the settings in the PG-FP5 flash programmer. Use command "ep" (erase and program) to download the S19 (or S37) file to the device. After successful programming download via bus the S19 file that was created by the modifier with the config_pgfp4.xml file. Because this file contains only load controls the download will be finished very quickly. For the download a tool is necessary that is able to interpret the load controls (for example the "KNX Device Editor").

6 Release Notes for BIM M 13x firmware revision 2.00

6.1 Introduction

With firmware revision 2.00 for the Siemens Bus Interface Modules the stability of the operating system was improved. All applications that are developed with firmware revisions below 2.00 can still be used with firmware revision 2.00 without recompiling or any modifications.

6.2 Improved Operating System

6.2.1 Timer

In firmware revisions below 2.00 it is possible that the function 'GetSystemTime' returns once a system ticks value that is about 106ms in the past. An immediate second call of this function will return the correct value. With firmware revision 2.00 the function 'GetSystemTime' always returns the correct value.
Application programs for firmware revisions below 2.00 should write the following statement in the first line of 'AppInit()' to prevent the described behavior above:

```
CR001 = 0xFFFF;
```

6.2.2 TPUART-Communication

It is rarely possible that the communication between the microcontroller with firmware revisions below 2.00 and the TPUART will stall in transmit way (microcontroller to TPUART). A failsafe function resets the communication and the system works again correctly.
In firmware revision 2.00 the communication between the microcontroller and the TPUART was improved to avoid this possibility of this stall of transmit communication.

6.2.3 FT12-Communication

All changes of the "M13x_FT12_Patch" and "M13x_FT12_Patch_v2" (necessary for firmware revisions below 2.00) are included in firmware revision 2.00. That means with firmware revision 2.00 the "M13x_FT12_Patch" and "M13x_FT12_Patch_v2" are no more necessary. Applications with one of these patches will still work correctly in firmware revision 2.00.