

# FlexRay Transfer Unit (FTU) Setup

Peter Steffan

## ABSTRACT

This application report summarizes the necessary steps to setup the FlexRay Transfer Unit (FTU) to transfer data between the FlexRay Message RAM and the data RAM of the microcontroller. Detailed code examples are provided as guidelines for the different setup steps.

<b>Acronym</b>	<b>Description</b>
CC	Communication Controller
FTU	FlexRay Transfer Unit
IBF	Input Buffer Register Set of the FlexRay Core (Eray)
OBF	Output Buffer Register Set of the FlexRay Core (Eray)
IBFS	Input Buffer Shadow Register Set of the FlexRay Core (Eray)
OBFS	Output Buffer Shadow Register Set of the FlexRay Core (Eray)
SM	System Memory (or Data RAM)
TCR	Transfer Configuration RAM

## Contents

1	Basic Steps to Setup an FTU Transfer .....	2
2	FTU Data Transfers .....	11
3	Code Examples .....	14
Appendix A	Listing of Further Useful Functions .....	19
Appendix B	Listing of FlexRay Message Header Structure .....	21
Appendix C	Listing of FTU Register Structure .....	23

## List of Figures

1	Message RAM Configuration Example.....	2
2	Data Package Storage Order in Data RAM .....	3
3	FTU Data Addressing .....	5
4	FlexRay Message Buffer Address Calculation Example.....	7
5	FTU Read Transfer of 6 Words.....	12
6	FTU Write Transfer of 6 Words.....	13

## List of Tables

1	Header Information Read Location .....	4
2	FTU Event Trigger Generation .....	13

## 1 Basic Steps to Setup an FTU Transfer

In order to use the FTU for data transfer between the FlexRay message RAM and the device data RAM, the module needs to be setup properly. The following bullets show the different basic steps that need to be performed to do this. Each item is discussed in detail in the following sections of this document.

- Proper configuration of the FlexRay message RAM
- Setup proper data buffer structure in the device System Memory (SM)
- Initialize FlexRay message buffers
- Define valid data RAM address space for FTU transfer
- Set data transfer base address
- Setup transfer direction in the configuration RAM (TCR)
- Enable FTU transfer interrupts (optional)
- Enable FTU
- Trigger FTU transfers
- Get transfer ready notification

### 1.1 FlexRay Message RAM Configuration

The header configuration of the FlexRay buffers in the FlexRay message RAM can be done through the conventional Input Buffer Register (IBF)/Output Buffer Register (OBF) set of the FlexRay module or by using the FTU.

Details about the configuration of the FlexRay Message RAM can be found in the *E-Ray FlexRay IP Module Application Note -- AN003 Message RAM Configuration* from Bosch. This document can be downloaded from the Bosch FlexRay IP webpage:

(<http://www.semiconductors.bosch.de/en/ipmodules/flexray/flexray.asp>).

In order to use the FTU to configure the FlexRay message RAM, a proper data buffer structure has to be setup in the device system memory, as defined in Section 1.2. The header portion of the message buffers can be transferred to the FlexRay message RAM through the FTU as shown in Figure 1.

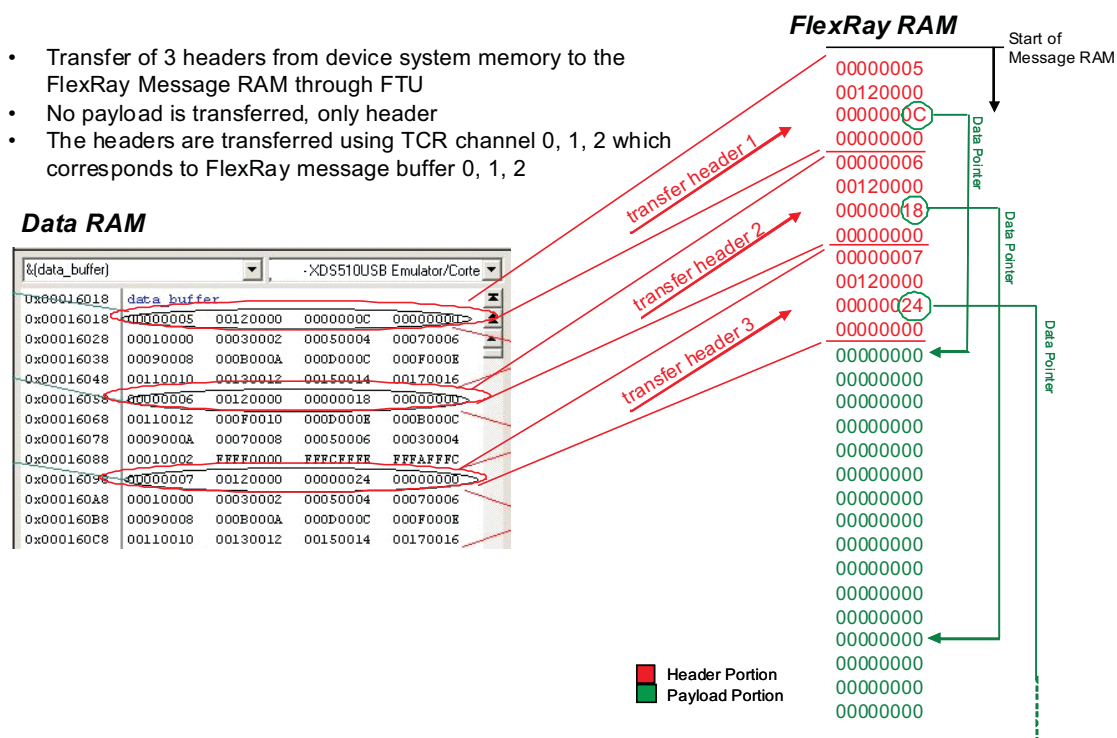


Figure 1. Message RAM Configuration Example

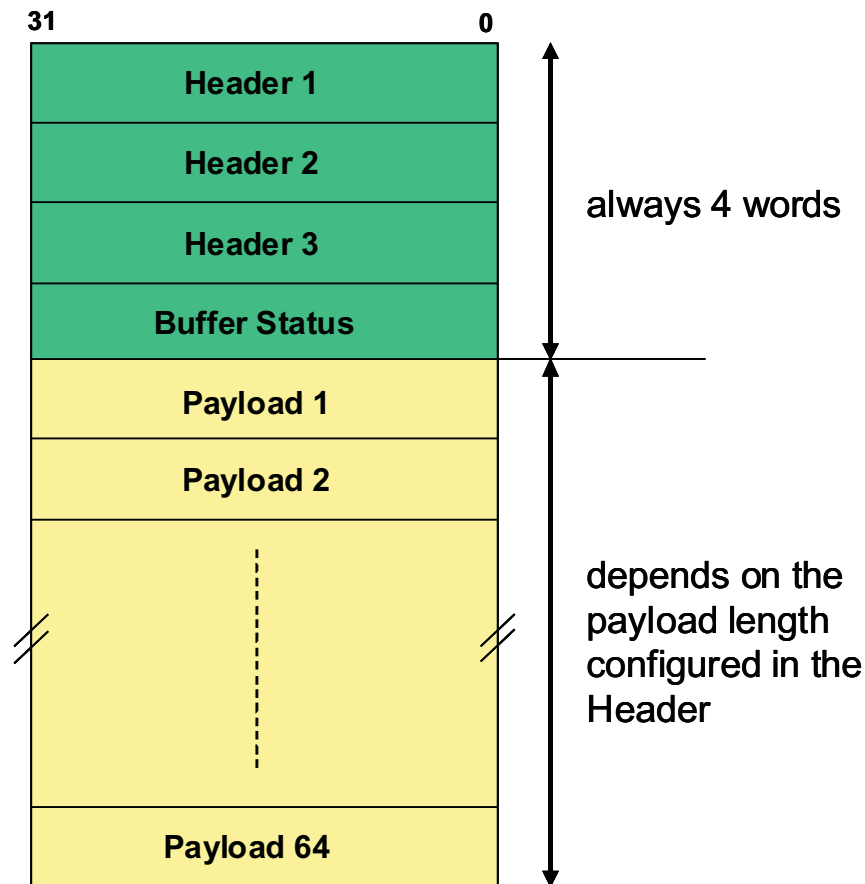
### 1.2 Setup Data Buffer Structure in the Device System Memory

In order to setup a proper data buffer structure in the data RAM, the following structure needs to be defined:

- 4 words for the Header information
- X halfwords for the Payload information

The length of payload X is identical for all FlexRay message buffers in the static segment. For FlexRay messages in the dynamic segment, this value can vary from buffer to buffer. The exact numbers are dependent on the FlexRay bus system setup and are typically defined during the FlexRay bus system definition phase.

For each FlexRay message buffer to be transferred by the FTU, the following data structure must be allocated in the data RAM:



**Figure 2. Data Package Storage Order in Data RAM**

Header segments are always 4 words, 3 header words and 1 buffer status word. The buffer status is transferred only from the FlexRay communication controller to the system memory, usually as status information of a received frame. However, the buffer status word must be considered in the data structure definition, whether this information is used or not.

The following code sequence shows the data structure of a FlexRay message buffer to be allocated in the data RAM. For details about the used FlexRay Message Header Structure (FRAY\_HEADER), see [Appendix A](#).

```
typedef volatile struct
{
    FRAY_RAM_ST FRAY_HEADER;
    unsigned int FRAY_DATA[FLEX_PAYLOAD];
}FLEX_MSG;
```

**NOTE:** Since the minimum data package of an FTU transfer is a burst of 4 words (4x32 bit), it must be ensured that the payload size of the data structure ends on a 4 word boundary.

### 1.3 Initialize FlexRay Message Buffers

Before starting any FTU transfer, it is essential that at least the header information is setup properly. Depending on the transfer direction, the FTU reads the header information either from the FlexRay message buffer of the FlexRay communication controller or from the data buffer structure in the system memory.

**Table 1. Header Information Read Location**

Transfer Type	Transfer Direction	Header Read Location
Header + Data	SM to CC	SM
Header + Data	CC to SM	CC
Header only	SM to CC	SM
Header only	CC to SM	CC
Data only	SM to CC	SM
Data only	CC to SM	CC

The header should contain at least the following information:

- Frame ID
- Data Pointer
- Payload Length

```
int ConfFlexHdr(FLEX_MSG * msg, T_U16 DataPtr, T_U16 FrameID, T_U8 length)
{
    if ((FrameID > 2048) | (DataPtr > 2048) | (length > 127))
    {
        return 1;
    }

    msg->FRAY_HEADER.HEADER1_UN.HEADER1_ST.FrameID = FrameID;
    msg->FRAY_HEADER.HEADER3_UN.HEADER3_ST.Data_Ptr = DataPtr;
    msg->FRAY_HEADER.HEADER2_UN.HEADER2_ST.PLC = length;

    return 0;
}
```

## 1.4 Define Valid Data RAM Address Space for FTU Transfer

After reset, the whole address space is protected against FTU accesses. Therefore, a dedicated data RAM address space must be defined before the first FTU transfer. One memory region can be defined by the FTU registers: Start Address of Memory Protection (SAMP) and the End Address of Memory Protection (EAMP), which allows read and write accesses for the FTU.

```
// Data RAM accessible from 0x08001000 to 0x08002000
FTU_Ptr->SAMP_UL = 0x08001000; // start address no memory protection for FTU
FTU_Ptr->EAMP_UL = 0x08002000; // end address no memory protection for FTU
```

## 1.5 Set Data Transfer Base Address

The data transfer base address defined in the Transfer Base Address Register (TBA) is a 32-bit pointer that points to the start address of the data buffer structure in the device system memory.

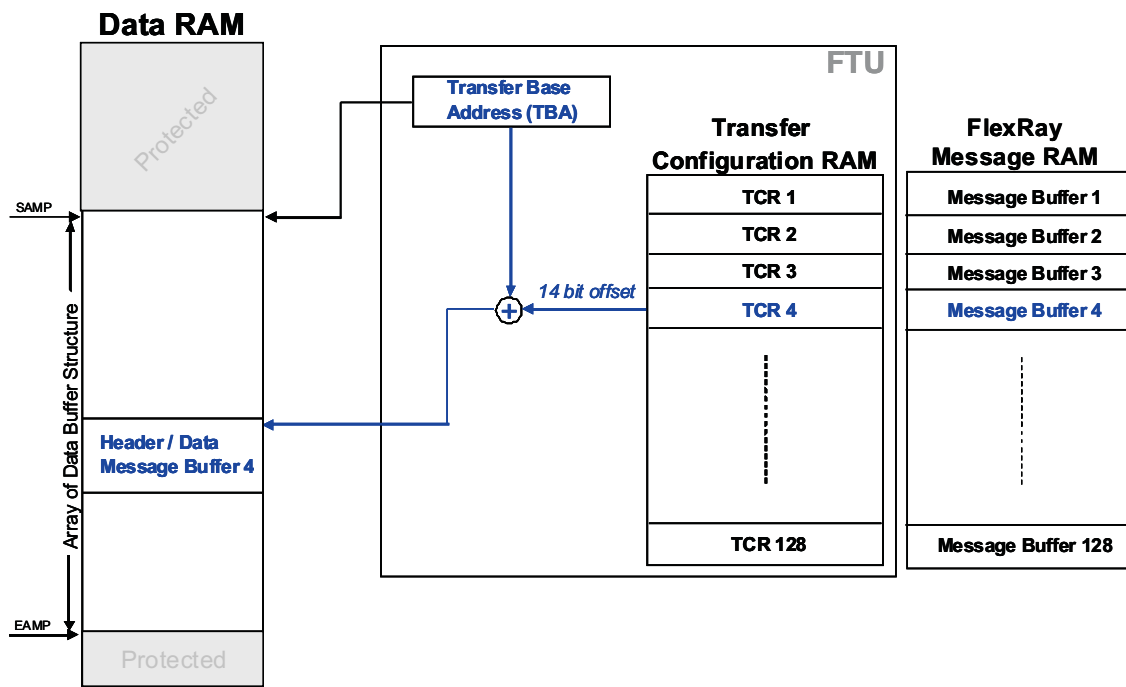
```
int FTU_TBA(FLEX_MSG * frame)
{
    // set source base address
    FTU_Ptr->TBA_UL = (unsigned long)frame;

    return 0;
}
```

## 1.6 Setup Transfer Configuration RAM

The transfer configuration RAM (TCR) consists of 128 entries. Each message buffer in the FlexRay message buffer RAM has one TCR entry assigned to it, i.e., message buffer 1 is assigned to TCR1, message buffer 2 is assigned to TCR2 and so on.

Figure 3 shows an example for FTU data addressing for message buffer 4.



**Figure 3. FTU Data Addressing**

Each TCR entry defines:

- The direction of the FTU transfer (SM to CC or CC to SM)
- Data transfer size (header + data, header only or data only)
- If the data, which has been transferred to the CC, should be sent out to the FlexRay bus by the FlexRay communication controller
- The 14-bit buffer address offset, which, in combination with the base address defined in the TBA register, specifies the start of the according FlexRay message buffer in the device system memory.

```

/*-----*/
/*          TCR transfer initialization          */
/* int TCR_Init(int channel, int header, int payload, int offset, int dir) */
/*          */
/* int channel:FTU channel                    */
/* int header:          1: transfers header    */
/*                   0: no header transfer    */
/* int payload:        1: transfers payload    */
/*                   0: no payload transfer   */
/* int offset:         transfer start offset   */
/* int dir:            1: transfer direction is to SM */
/*                   0: transfer direction is to CC */
/*          */
/* returns 0 */
/*-----*/

int TCR_Init(int channel, int header, int payload, int offset, int dir)
{
    int value = 0;

    if (dir != 0) // if direction is other than to CC
        dir = 2; // direction will be SM

    value = value + (header << (15 + dir));
    value = value + (payload << (14 + dir));
    value = value + offset;

    TCR_Ptr->FTUTCR_ST[channel].TCR_UN.TCR_UL = value;

    return(0);
}
    
```

**NOTE:** It is recommended to clear the whole TCR before configuration, in order to avoid unexpected transfer behavior due to old configuration contents or random TCR RAM contents after power on reset.

### 1.6.1 FTU Address Calculation of a FlexRay Message Buffer in the System RAM

The resulting address in system RAM is computed by adding the 32-bit aligned buffer address offset (TSO[13:0] = buffer address offset bits 15:2 in TCR) to the base address defined in the TBA register. In other words, the 14-bit buffer address offset defined in the TCR will be multiplied by 4, before adding it to the TBA value.

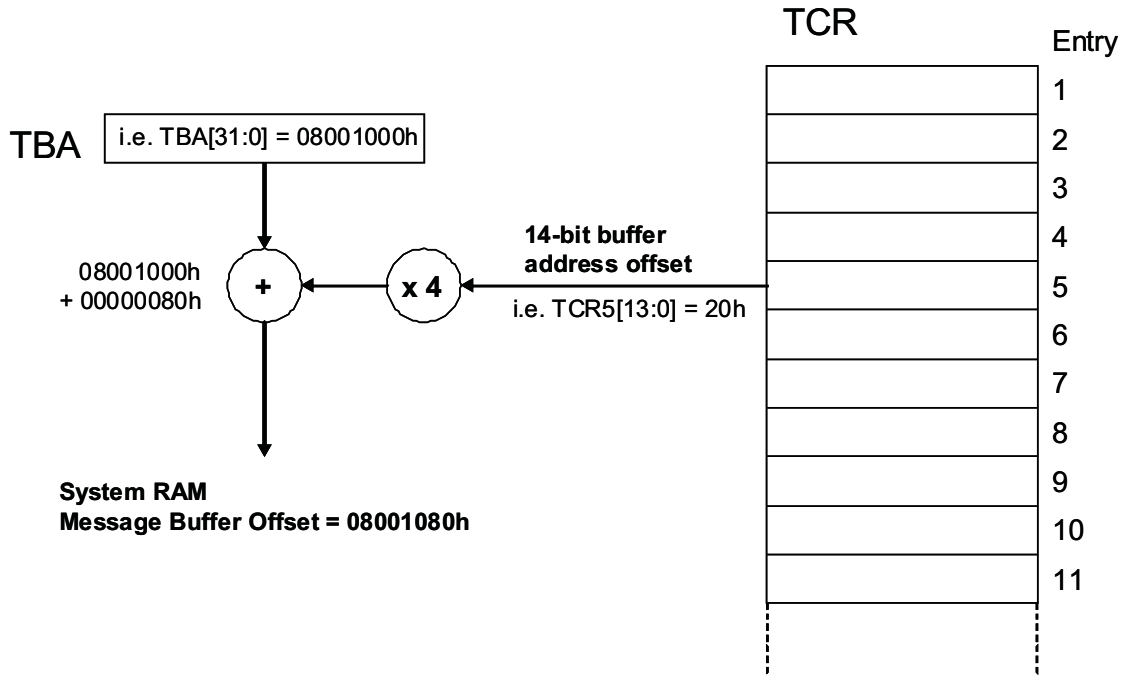


Figure 4. FlexRay Message Buffer Address Calculation Example

### 1.7 Enable FTU Transfer Interrupts

In order to get an interrupt notification when an FTU transfer sequence is finished, transfer interrupts can be generated. For each of the 128 FlexRay message buffers a transfer interrupt can be generated. There are two sets of four registers each: a Transfer to the System Memory Interrupt Enable Register (TSMIEx) set and a Transfer to Communication Controller Interrupt Enable Register (TCCIEEx) set, to enable dedicated interrupts.

This gives you high flexibility in generating interrupts according to your needs, i.e.,:

- After each transfer
- After a dedicated transfer is finished
- After each or dedicated transfers to SM
- After each or dedicated transfers to CC
- At the end of a transfer sequence

An interrupt can be generated after the transmission of a certain message buffer to indicate the completion of a transfer sequence. The FTU transfers message buffers from low buffer number to high buffer number.

The following code sequences show the setup for interrupts in both directions to system memory and communication controller:

```
int FTU_Multiple_Interrupt_TSMIE(int TSMIE0val, int TSMIE1val, int TSMIE2val, int TSMIE3val)
{
    // transfer to SM
    TSMIES_Ptr[0] = TSMIE0val;
    TSMIES_Ptr[2] = TSMIE1val;
    TSMIES_Ptr[4] = TSMIE2val;
    TSMIES_Ptr[6] = TSMIE3val;

    return 0;
}
```

```
int FTU_Multiple_Interrupt_TCCIE(int TCCIE0val, int TCCIE1val, int TCCIE2val, int TCCIE3val)
{
    // transfer to CC
    FTU_Ptr->TCCIES1_UL = TCCIE0val;
    FTU_Ptr->TCCIES2_UL = TCCIE1val;
    FTU_Ptr->TCCIES3_UL = TCCIE2val;
    FTU_Ptr->TCCIES4_UL = TCCIE3val;
    return 0;
}
```

---

**NOTE:** When using interrupts, it has to be ensured that the according interrupt line is enabled in the Global Control Register (GC) of the FTU. In addition, the global device interrupt generation must be setup and enabled properly. For more information about this topic, see the device-specific data sheet.

---

## 1.8 Enabling the FTU

The FTU is disabled after reset by default. During module initialization or re-initialization, the FTU should be disabled by the transfer unit enable bit in the global control register in order to avoid unexpected behaviors.

Disabling the FTU holds the transfer unit state machine in transfer idle state, but does not reset the module register's contents or the TCR. So, after re-enabling the TU, no further re-configuration is required.

After complete initialization, the FTU should be enabled by the transfer unit enable bit in the global control register.

```
void TU_enable()
{
    FTU_Ptr->GCS_UN.GCS_UL = TU_ENA; // enable FTU
}
```

## 1.9 Trigger FlexRay Buffer Transfers

Each of the 128 FlexRay buffers can be configured to be transferred individually. The transfer can be triggered either by the CPU or event triggered.

### 1.9.1 Transfer Trigger by CPU

Buffer transfers can be triggered by the CPU in both directions, from FlexRay message RAM of the FlexRay communication controller (CC) to the system memory (SM) of the microcontroller and vice versa, by using the Trigger Transfer to Communication Controller Register (TTCCx) and Trigger Transfer to System Memory Register (TTSMx) sets.



Each register set consists of 4 32-bit wide registers in order to represent all of the 128 possible FlexRay message buffers.

---

**NOTE:** The transfer triggered by TTSMx or TTCCx only occurs if the according transfer flags (THTSM, TPTSM, THTCC, TPTCC) are set in the TCR.

---

```
int FTU_Multiple_Trigger_TTCC(int TTCC0val, int TTCC1val, int TTCC2val, int TTCC3val)
{
    // transfer to buffer
    TTCC_Ptr[0] = TTCC0val;
    TTCC_Ptr[2] = TTCC1val;
    TTCC_Ptr[4] = TTCC2val;
    TTCC_Ptr[6] = TTCC3val;

    return 0;
}
```

```
. int FTU_Multiple_Trigger_TTSM(int TTSM0val, int TTSM1val, int TTSM2val, int TTSM3val)
{
    // transfer to buffer
    TTSM_Ptr[0] = TTSM0val;
    TTSM_Ptr[2] = TTSM1val;
    TTSM_Ptr[4] = TTSM2val;
    TTSM_Ptr[6] = TTSM3val;

    return 0;
}
```

A transfer of the same buffer in both directions is also possible. The transfer direction priority can be determined by the PRIO bit in the global control register. An example for the use of this feature is a data write to a buffer in the FlexRay message RAM and the read back of the same to check data constancy.

### 1.9.2 Event Triggered Transfer

Besides the manual trigger by the CPU, it is possible to transfer buffers from the FlexRay message RAM to the system memory 'on event'.

The general rule for the event triggering is: As soon as a FlexRay message is received by the FlexRay communication controller and has been stored in the FlexRay message RAM, the FTU is triggered to transfer this message buffer to the system memory. For more details concerning the FTU event trigger generation, see [Table 2](#).

Event driven buffer transfers can be enabled individually for all 128 FlexRay buffers by using the enable transfer on the Event to System Memory Registers (ETESMx).

```
int FTU_Multiple_Event_ETESMS(int ETESM0val, int ETESM1val, int ETESM2val, int ETESM3val)
{
    // transfer to SM
    ETESMS_Ptr[0] = ETESM0val;
    ETESMS_Ptr[2] = ETESM1val;
    ETESMS_Ptr[4] = ETESM2val;
    ETESMS_Ptr[6] = ETESM3val;

    return 0;
}
```

With the clear on the event to system memory registers (CESMx), it can be determined for each of the 128 buffers if the FTU transfer is continuous on every event or if the transfer stops after one event.

## 1.10 Transfer Complete Indication

There are two possible ways to indicate completed FTU transfers to the CPU:

- By generating an interrupt
- By polling transfer occurred flags

The most efficient way to indicate completed FTU transfers in terms of CPU load is to generate an interrupt, i.e., at the end of an FTU transfer sequence. [Section 1.7](#) describes the different options for transfer interrupt generation and how to set them up.

The second possibility of transfer complete indication is the Transfer to Communication Controller Occurred (TCCOx) and the Transfer to System Memory Occurred (TSMOx) registers. There are four registers of each type (TCCOx and TSMOx) to be able to indicate the transfer of each of the 128 buffers for both of the possible transfer directions. The flags of the TCCOx registers indicate completed transfers from the system memory to the FlexRay communication controller, whereas, the TSMOx flags show completed transfers from the FlexRay communication controller to the system memory.

The CPU can poll on a certain bit in order to see if a transfer sequence has been finished. The FTU transfer message buffers from low buffer number to high buffer number.

The following code sequences show polling for transfer completion:

```

/*-----*/
/* Multiple FTU Transfer to Communication Controller Completed */
/*          int FTU_Multiple_TCCC () */
/*          */
/* returns 1 if FTU transfers completed */
/*-----*/
int FTU_Multiple_TCCC(int TTCC0val, int TTCC1val, int TTCC2val, int TTCC3val)
{
    T_U32 value0, value1, value2, value3;

    value0 = FTU_Ptr->TCCO1_UL;
    value1 = FTU_Ptr->TCCO2_UL;
    value2 = FTU_Ptr->TCCO3_UL;
    value3 = FTU_Ptr->TCCO4_UL;

    if ((value0 == TTCC0val)&&(value1 == TTCC1val)&&(value2 == TTCC2val)&&(value3 == TTCC3val))
    {
        FTU_Ptr->TCCO1_UL = value0;
        FTU_Ptr->TCCO2_UL = value1;
        FTU_Ptr->TCCO3_UL = value2;
        FTU_Ptr->TCCO4_UL = value3;

        return(1);
    }
    else
    {
        return (0);
    }
}

```

```

/*-----*/
/* Multiple FTU Transfer to System Memory Completed */
/*          int FTU_Multiple_TCSM () */
/*          */
/* returns 1 if FTU transfers completed */
/*-----*/
int FTU_Multiple_TCSM(int TTSM0val, int TTSM1val, int TTSM2val, int TTSM3val)
{
    T_U32 value0, value1, value2, value3;

    value0 = FTU_Ptr->TSMO1_UL;

```

```

value1 = FTU_Ptr->TSMO2_UL;
value2 = FTU_Ptr->TSMO3_UL;
value3 = FTU_Ptr->TSMO4_UL;

if ((value0 == TTSM0val)&&(value1 == TTSM1val)&&(value2 == TTSM2val)&&(value3 == TTSM3val))
{
FTU_Ptr->TSMO1_UL = value0;
FTU_Ptr->TSMO2_UL = value1;
FTU_Ptr->TSMO3_UL = value2;
FTU_Ptr->TSMO4_UL = value3;

return(1);
}
else
{
return (0);
}

```

## 2 FTU Data Transfers

The following section describes how the FTU data transfer works.

### 2.1 Transfer Options

A data buffer can be divided into two sections: the header and payload. The sections of the message buffer that get transferred by the FTU can be configured in the TCR.

- Header and payload
- Header only
- Payload only

The size of the header section is always 4 words, whereas, the size of the payload section can vary from 0 to 127 half-words (254 bytes). The size of the payload is defined in the buffers header section and is read by the FTU from the same location. Therefore, the payload length must be defined in the header partition before an FTU transfer can be started, even if the header section is not part of the FTU transfer (see [Table 2](#) for details).

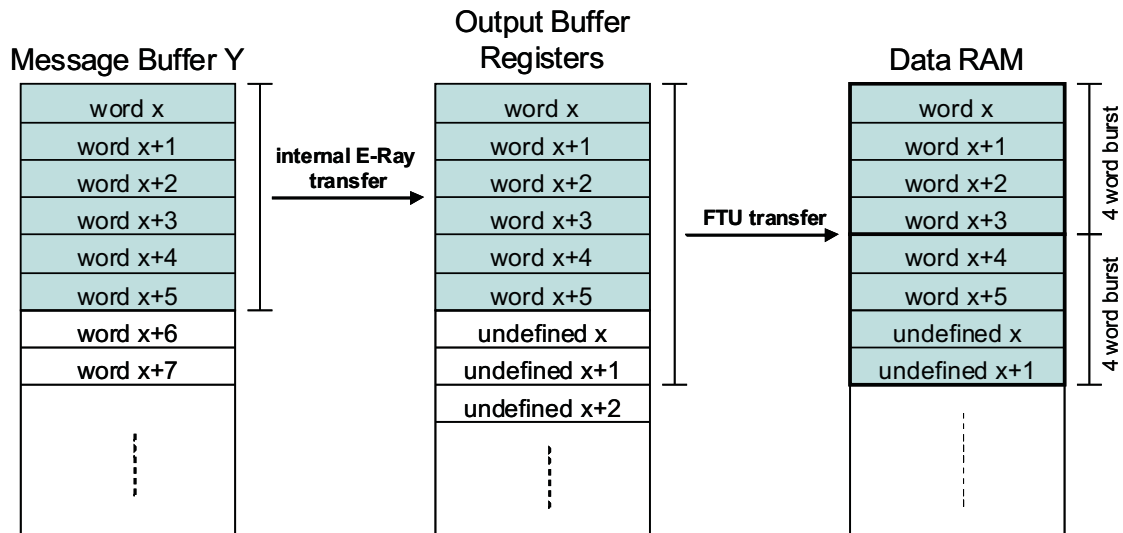
#### 2.1.1 Header and Payload Versus Payload Only Transfer

One special case is the 'payload only' transfer from the FlexRay message RAM to the system memory. Even if the header information of 4 words are not transferred in this mode; the FTU has to read the header information from the message buffer header in order to determine the payload length of the data to be transferred. Therefore, the 'payload only' transfer does not gain much transfer time performance versus the transfer with header and payload.

### 2.2 Transfer Bursts

The FTU always transfers 4 words aligned, or in other words, in 4 word bursts. Therefore, it makes no difference, if the payload size is, i.e., 2, 3 or 4 words. The FTU always transfers 4 word aligned data packages and fills the additional words to the next 4 word boundary with 0.

Figure 5 shows a 6 word FTU transfer from FlexRay message RAM to data RAM.



**Figure 5. FTU Read Transfer of 6 Words**

The transfer can be divided in two parts: the internal E-Ray transfer and the real FTU transfer.

During the internal E-Ray transfer, the requested data are copied to the E-Ray output buffer register set by the E-Ray internal message handler state machine. The subsequent FTU transfer copies the data from the output buffer register set (OBF) to the data RAM of the microcontroller.

In the example above, 6 data words of 32 bit size are transferred. After the FTU requests the 6 data words of Message Buffer Y, the E-Ray internal message handler state machine copies this information from the FlexRay message buffer to the output buffer register set of the E-Ray to make the data readable for the FTU.

In the second step, the FTU starts to copy the data from the output buffer register set to the data RAM location defined by the TBA register and the TCR entry contents (see Figure 2 for more details). Since the FTU is transferring 4 word bursts, finally 8 data words get copied from the output buffer register set to the data RAM. The two additional copied data words have undefined contents and need to be ignored.

Timing wise there are two components: the E-Ray internal transfer time from the message buffer to the output buffer register set and the FTU transfer itself. If more than 1 message buffer transfer is triggered, the FTU requests the subsequent message buffer data from the E-Ray, while copying the data from the output buffer register set (OBF) to the system memory. With this, the E-Ray internal message handler state machine can copy the next data to a output buffer shadow register set (OBFS), while the FTU transfer of the current message buffer data is ongoing. This parallel data copy mechanism ensures optimized FTU transfer performance.

The FTU transfer from the system memory to the FlexRay message RAM basically works the same way using the input buffer register set (IBF) and a input buffer shadow register set (IBFS). Table 2 shows that FTU overwrites two additional words in the input buffer register set, but does not transfer the additional words further to the FlexRay message buffer Y.

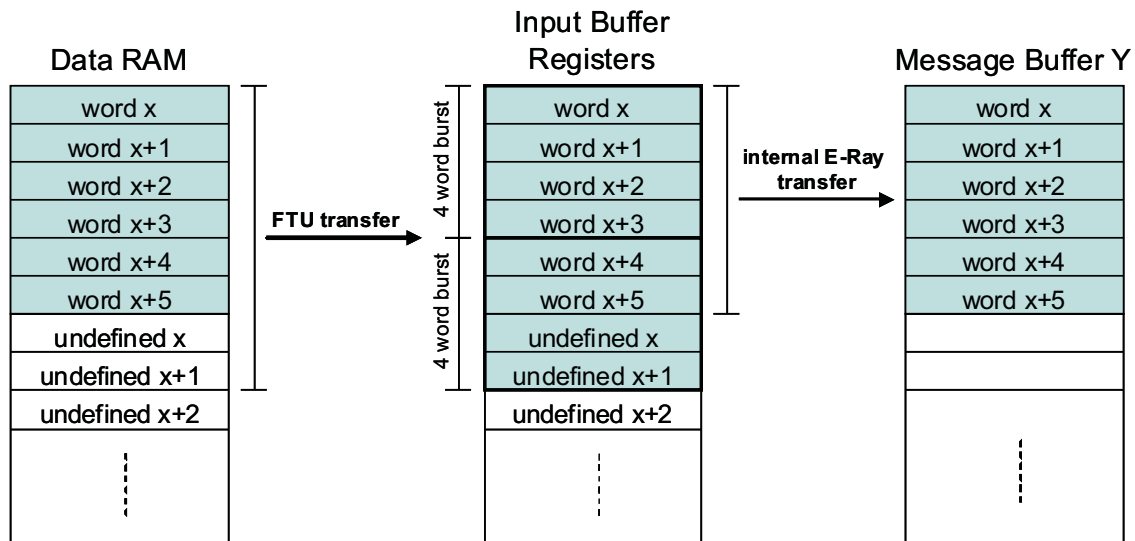


Figure 6. FTU Write Transfer of 6 Words

### 2.3 Transfer of NULL Frames and Empty Frames

Table 2 summarizes the exact conditions the FTU triggers an event.

Table 2. FTU Event Trigger Generation <sup>(1)</sup>

	Event on Channel A	Event on Channel B	Frame Belonging to Static Segment or First Slot of Dynamic Segment	Frame Belonging to Dynamic Segment, Except First Slot of Dynamic Segment	Bus Activity Detected on Channel A (MBS.ESA = 0)	Bus Activity Detected on Channel B (MBS.ESB = 0)
FTU Event Trigger for Receive Message Buffers	X		X			
		X	X			
	X			X	X	
		X		X		X
FTU Event Trigger for Transmit Message Buffers	X				X	
		X				X

<sup>(1)</sup> All conditions per row, marked with 'X,' must match to trigger an FTU transfer.

The handling of NULL frames and empty frames by the FTU can be summarized as:

- For a received message, the buffer in the static segment NULL frames or empty frames generate an FTU trigger. It depends on the settings in the TCR whether the header and/or payload get transferred by the FTU. The corresponding bit in the Transfer to System Memory Occurred Register (TSMO) gets set in all cases, even if neither header nor payload get transferred.
- No event trigger is generated for an empty receive message in the dynamic segment.

### 3 Code Examples

#### 3.1 Transfer Message Data Structures in Polling Mode

##### 3.1.1 Transfer to System Memory (Polling)

```
//===== Transfer Message Data Structures to FlexRay Message RAM =====

// disable FTU
TU_disable();
//Clear Event to System Memory register
TU_ETESM_clear();
//Clear Trigger Transfer to Communication Controller register
TU_TTCC_clear();
//Clear Trigger Transfer to System Memory register
TU_TTSM_clear();

// setup Transfer Config RAM for transfer to E-Ray message buffer
for (y=0; y< FLEX_BUFFERS ;y++)
    //int TCR_Init(int channel, int header, int payload, int offset, int dir)
    TCR_Init(y, 1, 1, y*(FLEX_PAYLOAD_32 + HDR_LENGTH), SM2CC);

// set data buffer base address
FTU_TBA(data_buffer);

// enable FTU
TU_enable();

// trigger multiple transfers
//int FTU_Multiple_Trigger_TTCC(int TTCC0val, int TTCC1val, int TTCC2val, int TTCC3val)
    FTU_Multiple_Trigger_TTCC(0xFFFFFFFF,0xFFFFFFFF,0xFFFFFFFF,0xFFFFFFFF);

// wait for multiple TCCC (Transfer to CC Completed)
    while (FTU_Multiple_TCCC(0xFFFFFFFF,0xFFFFFFFF,0xFFFFFFFF,0xFFFFFFFF) == 0);

//=====
```

##### 3.1.2 Transfer in both Directions (Polling)

```
//===== Simultaneously transfer in both directions =====

// disable FTU
TU_disable();
//Clear Event to System Memory register
TU_ETESM_clear();
//Clear Trigger Transfer to Communication Controller register
TU_TTCC_clear();
//Clear Trigger Transfer to System Memory register
TU_TTSM_clear();

// setup Transfer Config RAM for transfer to E-Ray message buffer
for (y=0; y< FLEX_BUFFERS/2 ;y++)
    //int TCR_Init(int channel, int header, int payload, int offset, int dir)
    TCR_Init(y, 1, 1, y*(FLEX_PAYLOAD_32 + HDR_LENGTH), SM2CC);

for (y=FLEX_BUFFERS/2; y< FLEX_BUFFERS ;y++)
    //int TCR_Init(int channel, int header, int payload, int offset, int dir)
    TCR_Init(y, 1, 1, y*(FLEX_PAYLOAD_32 + HDR_LENGTH), CC2SM);

// trigger multiple transfers
//int FTU_Multiple_Trigger_TTCC(int TTCC0val, int TTCC1val, int TTCC2val, int TTCC3val)
```

```

FTU_Multiple_Trigger_TTCC(0xFFFFFFFF,0xFFFFFFFF,0,0);

// trigger multiple transfers
//int FTU_Multiple_Trigger_TTSM(int TTSM0val, int TTSM1val, int TTSM2val, int TTSM3val)
  FTU_Multiple_Trigger_TTSM(0,0,0xFFFFFFFF,0xFFFFFFFF);

// set data buffer base address
FTU_TBA(data_rx_buffer);

// enable FTU
TU_enable();

```

## 3.2 Transfer Message Data Structures in Interrupt Mode

### 3.2.1 Enable FTU Interrupts

```

// enable IRQ and FIQ
irq_fiq_enable();

// enable FTU int0 interrupt line
ftu_int0_enable();
ftu_int1_enable();

```

### 3.2.2 Transfer to Communication Controller (Interrupt)

```

//===== Transfer Message Data Structures to FlexRay Message RAM =====

// disable FTU
TU_disable();
//Clear Event to System Memory register
TU_ETESM_clear();
//Clear Trigger Transfer to Communication Controller register
TU_TTCC_clear();
//Clear Trigger Transfer to System Memory register
TU_TTSM_clear();

// setup Transfer Config RAM for transfer to E-Ray message buffer
for (y=0; y< FLEX_BUFFERS ;y++)
    //int TCR_Init(int channel, int header, int payload, int offset, int dir)
    TCR_Init(y, 1, 1, y*(FLEX_PAYLOAD_32 + HDR_LENGTH), SM2CC);

// trigger multiple transfers
//int FTU_Multiple_Trigger_TTCC(int TTCC0val, int TTCC1val, int TTCC2val, int TTCC3val)
  FTU_Multiple_Trigger_TTCC(0xFFFFFFFF,0xFFFFFFFF,0xFFFFFFFF,0xFFFFFFFF);

// Transfer to CC interrupt enable
FTU_Multiple_Interrupt_TCCIE(0, 0, 0, 0x80000000);

// set data buffer base address
FTU_TBA(data_buffer);

// enable FTU
TU_enable();

```

### 3.2.3 Transfer to System Memory (Interrupt)

```
//===== Transfer Message Data Structures to Receive Data Buffer =====

// disable FTU
TU_disable();
//Clear Event to System Memory register
TU_ETESM_clear();
//Clear Trigger Transfer to Communication Controller register
TU_TTCC_clear();
//Clear Trigger Transfer to System Memory register
TU_TTSM_clear();

for (y=0; y< FLEX_BUFFERS ;y++)
    //int TCR_Init(int channel, int header, int payload, int offset, int dir)
    TCR_Init(y, 1, 1, y*(FLEX_PAYLOAD_32 + HDR_LENGTH), CC2SM);

// trigger multiple transfers
//int FTU_Multiple_Trigger_TTSM(int TTSM0val, int TTSM1val, int TTSM2val, int TTSM3val)
    FTU_Multiple_Trigger_TTSM(0xFFFFFFFF,0xFFFFFFFF,0xFFFFFFFF,0xFFFFFFFF);

// Transfer to SM interrupt enable
FTU_Multiple_Interrupt_TSMIE(0, 0, 0, 0x80000000);

// set data buffer base address
FTU_TBA(data_rx_buffer);

// enable FTU
TU_enable();
```

### 3.2.4 Transfer in Both Directions (Interrupt)

```
//===== Simultaneously transfer in both directions =====

// disable FTU
TU_disable();
//Clear Event to System Memory register
TU_ETESM_clear();
//Clear Trigger Transfer to Communication Controller register
TU_TTCC_clear();
//Clear Trigger Transfer to System Memory register
TU_TTSM_clear();

// setup Transfer Config RAM for transfer to E-Ray message buffer
for (y=0; y< FLEX_BUFFERS/2 ;y++)
    //int TCR_Init(int channel, int header, int payload, int offset, int dir)
    TCR_Init(y, 1, 1, y*(FLEX_PAYLOAD_32 + HDR_LENGTH), SM2CC);

for (y=FLEX_BUFFERS/2; y< FLEX_BUFFERS ;y++)
    //int TCR_Init(int channel, int header, int payload, int offset, int dir)
    TCR_Init(y, 1, 1, y*(FLEX_PAYLOAD_32 + HDR_LENGTH), CC2SM);

// trigger multiple transfers
//int FTU_Multiple_Trigger_TTCC(int TTCC0val, int TTCC1val, int TTCC2val, int TTCC3val)
    FTU_Multiple_Trigger_TTCC(0xFFFFFFFF,0xFFFFFFFF,0,0);

// trigger multiple transfers
//int FTU_Multiple_Trigger_TTSM(int TTSM0val, int TTSM1val, int TTSM2val, int TTSM3val)
    FTU_Multiple_Trigger_TTSM(0,0,0xFFFFFFFF,0xFFFFFFFF);

// Interrupt when last transfer to SM interrupt enable
FTU_Multiple_Interrupt_TSMIE(0, 0, 0, 0x80000000);
```



```
// set data buffer base address
FTU_TBA(data_rx_buffer);

// enable FTU
```

### 3.2.5 Interrupt Enable and Handler Routines

```
void ftu_int0_enable()
{
    FTU_Ptr->GCS_UN.GCS_UL = INT0_ENA;// enable int0
}

void ftu_int1_enable()
{
    FTU_Ptr->GCS_UN.GCS_UL = INT1_ENA;// enable int1
}

void handle_FTU_int0(){

    tu_busy = 1;
    // clear all pending TSMO bits
    FTU_Ptr->TSMO1_UL = 0xffffffff;
    FTU_Ptr->TSMO2_UL = 0xffffffff;
    FTU_Ptr->TSMO3_UL = 0xffffffff;
    FTU_Ptr->TSMO4_UL = 0xffffffff;

    // clear all pending TCCO bits
    FTU_Ptr->TCCO1_UL = 0xffffffff;
    FTU_Ptr->TCCO2_UL = 0xffffffff;
    FTU_Ptr->TCCO3_UL = 0xffffffff;
    FTU_Ptr->TCCO4_UL = 0xffffffff;

    // clear all pending TSMIE bits
    FTU_Ptr->TSMIER1_UL = 0xffffffff;
    FTU_Ptr->TSMIER2_UL = 0xffffffff;
    FTU_Ptr->TSMIER3_UL = 0xffffffff;
    FTU_Ptr->TSMIER4_UL = 0xffffffff;

    // clear all pending TCCIE bits
    FTU_Ptr->TCCIER1_UL = 0xffffffff;
    FTU_Ptr->TCCIER2_UL = 0xffffffff;
    FTU_Ptr->TCCIER3_UL = 0xffffffff;
    FTU_Ptr->TCCIER4_UL = 0xffffffff;
}
}
```

## 3.3 Event Driven Transfer of Message Data Structures

### 3.3.1 Event Triggered Transfer

```
//===== Transfer Message Data Structures to Receive Data Buffer =====

// disable FTU
TU_disable();
//Clear Event to System Memory register
TU_ETESM_clear();
//Clear Trigger Transfer to Communication Controller register
TU_TTCC_clear();
//Clear Trigger Transfer to System Memory register
TU_TTSM_clear();
```

```
for (y=0; y< FLEX_BUFFERS ;y++)
int TCR_Init(int channel, int header, int payload, int offset, int dir, int send)

// trigger multiple event transfers
FTU_Multiple_Event_ETESMS(0x000000f,0x00000000,0x00000000,0x00000000);

// Transfer to SM interrupt enable
//FTU_Multiple_Interrupt_TSMIE(0, 0, 0, 0x80000000);

// set data buffer base address
FTU_TBA(data_rx_buffer);

// enable FTU
TU_enable();
```

## Appendix A Listing of Further Useful Functions

```
/*-----*/
/*      Clear Event to System Memory register*/
/*      void TU_ETESM_clear()                */
/*-----*/
void TU_ETESM_clear()
{
    FTU_Ptr->GCS_UN.GCS_UL = 1<<14;// disable FTU
    FTU_Ptr->GCR_UN.GCR_UL = 1<<14;// disable FTU
}

/*-----*/
/*      Clear Trigger Transfer to Communication Controller register */
/*      void TU_TTCC_clear()                */
/*-----*/
void TU_TTCC_clear()
{
    FTU_Ptr->GCS_UN.GCS_UL = 1<<13;// disable FTU
    FTU_Ptr->GCR_UN.GCR_UL = 1<<13;// disable FTU
}

/*-----*/
/*      Clear Trigger Transfer to System Memory register */
/*      void TU_TTSM_clear()                */
/*-----*/
void TU_TTSM_clear()
{
    FTU_Ptr->GCS_UN.GCS_UL = 1<<12;// disable FTU
    FTU_Ptr->GCR_UN.GCR_UL = 1<<12;// disable FTU
}
```



## Appendix B Listing of FlexRay Message Header Structure

```

// FlexRay header in message RAM

typedef volatile struct fray_ram_header
{
    union h1
    {
        unsigned long HEADER1_UL;
        struct
        {
            unsigned          : 2;
            unsigned MBI      : 1;
            unsigned TXM      : 1;
            unsigned NME      : 1;
            unsigned CFG      : 1;
            unsigned CHB      : 1;
            unsigned CHA      : 1;
            unsigned          : 1;
            unsigned Cycle_Code : 7;
            unsigned          : 5;
            unsigned FrameID   : 11;
        } HEADER1_ST;
    } HEADER1_UN;

    union h2
    {
        unsigned long HEADER2_UL;
        struct
        {
            unsigned          : 1;
            unsigned PLR      : 7;
            unsigned          : 1;
            unsigned PLC      : 7;
            unsigned          : 5;
            unsigned crc      : 11;
        } HEADER2_ST;
    } HEADER2_UN;

    union h3
    {
        unsigned long HEADER3_UL;
        struct
        {
            unsigned          : 2;
            unsigned RES      : 1;
            unsigned PPI      : 1;
            unsigned NFI      : 1;
            unsigned SYN      : 1;
            unsigned SFI      : 1;
            unsigned RCI      : 1;
            unsigned          : 2;
            unsigned RCcnt    : 6;
            unsigned          : 5;
            unsigned Data_Ptr : 11;
        } HEADER3_ST;
    } HEADER3_UN;

    union h4
    {
        unsigned long HEADER4_UL;
        struct
        {

```

```
unsigned          : 19;
unsigned MLST     : 1;
unsigned ESB      : 1;
unsigned ESA      : 1;
unsigned TCIB     : 1;
unsigned TCIA     : 1;
unsigned SVOB     : 1;
unsigned SVOA     : 1;
unsigned CEOB     : 1;
unsigned CEOA     : 1;
unsigned SEOB     : 1;
unsigned SEOA     : 1;
unsigned VFRB     : 1;
unsigned VFRA1    : 1;
} HEADER4_ST;
} HEADER4_UN;
}FRAY_RAM_ST;
```

## Appendix C Listing of FTU Register Structure

```

typedef volatile struct
{
    unsigned long   PICC_UL;

    unsigned long   GSN_UL;

    unsigned int:32;
    unsigned int:32;

    union
    {
        {
            unsigned long GCS_UL;
            struct
            {
                unsigned intENDVBM_B1 :1;
                unsigned intENDVBS_B1 :1;
                unsigned intENDRx_B2  :2;
                unsigned intENDHx_B2  :2;
                unsigned intENDPx_B2  :2;
                unsigned int   :2;
                unsigned intPROI_B1   :1;
                unsigned intPFT_B1    :1;
                unsigned intPALx_B2   :1;
                unsigned int   :1;
                unsigned intCETESM_B1 :1;
                unsigned intCTTCC_B1  :1;
                unsigned intCTTSM_B1  :1;
                unsigned int   :2;
                unsigned intETSM_B1   :1;
                unsigned int   :2;
                unsigned intSILE_B1   :1;
                unsigned intEILE_B1   :1;
                unsigned int   :2;
                unsigned intTUH_B1    :1;
                unsigned intTUE_B1    :1;
            } GCS_ST;
        } GCS_UN;

        union
        {
            unsigned long GCR_UL;
            struct
            {
                unsigned intENDVBM_B1 :1;
                unsigned intENDVBS_B1 :1;
                unsigned intENDRx_B2  :2;
                unsigned intENDHx_B2  :2;
                unsigned intENDPx_B2  :2;
                unsigned int   :2;
                unsigned intPROI_B1   :1;
                unsigned intPFT_B1    :1;
                unsigned intPALx_B2   :1;
                unsigned int   :1;
                unsigned intCETESM_B1 :1;
                unsigned intCTTCC_B1  :1;
                unsigned intCTTSM_B1  :1;
                unsigned int   :2;
                unsigned intETSM_B1   :1;
                unsigned int   :2;
                unsigned intSILE_B1   :1;
                unsigned intEILE_B1   :1;
            }
        }
    }
}

```

```

    unsigned int      :2;
    unsigned intTUH_B1 :1;
    unsigned intTUE_B1 :1;
} GCR_ST;
} GCR_UN;

union
{
unsigned long TSCB_UL;
struct
{
    unsigned int      :11;
    unsigned intTSMS_B5 :5;
    unsigned int      :3;
    unsigned intSTUH_B1 :1;
    unsigned int      :3;
    unsigned intIDLE_B1 :1;
    unsigned int      :1;
    unsigned intBN_B7  :1;
} TSCB_ST ;
} TSCB_UN;
unsigned long LTBCC_UL;

unsigned long LTBSM_UL;

unsigned long TBA_UL;

unsigned long NTBA_UL;

unsigned long BAMS_UL;

unsigned long SAMP_UL;

unsigned long EAMP_UL;

unsigned int:32;
unsigned int:32;

unsigned long TSMO1_UL;
unsigned long TSMO2_UL;
unsigned long TSMO3_UL;
unsigned long TSMO4_UL;

unsigned long TCCO1_UL;
unsigned long TCCO2_UL;
unsigned long TCCO3_UL;
unsigned long TCCO4_UL;

union
{
unsigned long TOFF_UL;
struct
{
    unsigned int      : 23;
    unsigned int TDIR_B1 : 1;
    unsigned int OFF_B8  : 8;
}TOFF_ST ;
} TOFF_UN;
unsigned int:32;
unsigned int:32;
unsigned int:32;

unsigned long PEADR_UL;

union
{

```



```

unsigned long TEIR_UL;
struct
{
    unsigned int      : 14;
    unsigned int MPV_B1 : 1;
    unsigned int PE_B1  : 1;
    unsigned int      : 5;
    unsigned int RSTAT_B3 : 3;
    unsigned int      : 1;
    unsigned int SSTAT_B3 : 3;
    unsigned int      : 2;
    unsigned int TNR_B1  : 1;
    unsigned int FAC_B1  : 1;
}TEIR_ST ;
} TEIR_UN;

union
{
    unsigned long TEIRES_UL;
    struct
    {
        unsigned int      : 21;
        unsigned int RSTAT_B3 : 3;
        unsigned int      : 1;
        unsigned int SSTAT_B3 : 3;
        unsigned int      : 2;
        unsigned int TNR_B1  : 1;
        unsigned int FAC_B1  : 1;
    }TEIRES_ST ;
    } TEIRES_UN;

union
{
    unsigned long TEIRER_UL;
    struct
    {
        unsigned int      : 21;
        unsigned int RSTAT_B3 : 3;
        unsigned int      : 1;
        unsigned int SSTAT_B3 : 3;
        unsigned int      : 2;
        unsigned int TNR_B1  : 1;
        unsigned int FAC_B1  : 1;
    }TEIRER_ST ;
    } TEIRER_UN;

unsigned long TTSMS1_UL;
unsigned long TTSMR1_UL;
unsigned long TTSMS2_UL;
unsigned long TTSMR2_UL;
unsigned long TTSMS3_UL;
unsigned long TTSMR3_UL;
unsigned long TTSMS4_UL;
unsigned long TTSMR4_UL;

unsigned long TTCCS1_UL;
unsigned long TTCCR1_UL;
unsigned long TTCCS2_UL;
unsigned long TTCCR2_UL;
unsigned long TTCCS3_UL;
unsigned long TTCCR3_UL;
unsigned long TTCCS4_UL;
unsigned long TTCCR4_UL;

unsigned long ETESMS1_UL;
unsigned long ETESMR1_UL;

```

```
unsigned long ETESMS2_UL;  
unsigned long ETESMR2_UL;  
unsigned long ETESMS3_UL;  
unsigned long ETESMR3_UL;  
unsigned long ETESMS4_UL;  
unsigned long ETESMR4_UL;  
  
unsigned long CESMS1_UL;  
unsigned long CESMR1_UL;  
unsigned long CESMS2_UL;  
unsigned long CESMR2_UL;  
unsigned long CESMS3_UL;  
unsigned long CESMR3_UL;  
unsigned long CESMS4_UL;  
unsigned long CESMR4_UL;  
  
unsigned long TSMIES1_UL;  
unsigned long TSMIER1_UL;  
unsigned long TSMIES2_UL;  
unsigned long TSMIER2_UL;  
unsigned long TSMIES3_UL;  
unsigned long TSMIER3_UL;  
unsigned long TSMIES4_UL;  
unsigned long TSMIER4_UL;  
  
unsigned long TCCIES1_UL;  
unsigned long TCCIER1_UL;  
unsigned long TCCIES2_UL;  
unsigned long TCCIER2_UL;  
unsigned long TCCIES3_UL;  
unsigned long TCCIER3_UL;  
unsigned long TCCIES4_UL;  
unsigned long TCCIER4_UL;  
} FTU_ST;
```

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

### Products

Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>
DLP® Products	<a href="http://www.dlp.com">www.dlp.com</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>
Clocks and Timers	<a href="http://www.ti.com/clocks">www.ti.com/clocks</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>
OMAP Mobile Processors	<a href="http://www.ti.com/omap">www.ti.com/omap</a>
Wireless Connectivity	<a href="http://www.ti.com/wirelessconnectivity">www.ti.com/wirelessconnectivity</a>

### Applications

Automotive and Transportation	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Communications and Telecom	<a href="http://www.ti.com/communications">www.ti.com/communications</a>
Computers and Peripherals	<a href="http://www.ti.com/computers">www.ti.com/computers</a>
Consumer Electronics	<a href="http://www.ti.com/consumer-apps">www.ti.com/consumer-apps</a>
Energy and Lighting	<a href="http://www.ti.com/energy">www.ti.com/energy</a>
Industrial	<a href="http://www.ti.com/industrial">www.ti.com/industrial</a>
Medical	<a href="http://www.ti.com/medical">www.ti.com/medical</a>
Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Space, Avionics and Defense	<a href="http://www.ti.com/space-avionics-defense">www.ti.com/space-avionics-defense</a>
Video and Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>

TI E2E Community Home Page

[e2e.ti.com](http://e2e.ti.com)

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2012, Texas Instruments Incorporated