
TN8000.17

Technical Note

BitJockey™ basic driver

Table of Contents

1.	INTRODUCTION	3
2.	GENERAL GOAL OF THE DOCUMENT	4
3.	GLOBAL DESCRIPTION	5
4.	DRIVER CONTENT.....	6
5.	FUNCTIONS DESCRIPTION	6
5.1	ASM_INITIATE_TX.....	6
5.2	ASM_INITIATE_RX	7
5.3	HANDLE_IRQ_RFIF_TX.....	7
5.4	HANDLE_IRQ_RFIF_RX	8
6.	HOW TO USE THE BASIC DRIVER	8
7.	REFERENCE.....	9
8.	INITIATE_RF.S.....	9
9.	RF_IRQ_HANDLER.S	10

1. INTRODUCTION

The BitJockey™ is a PCM Bit Stream Encoder / Decoder or RF Receiver / Transmitter Interface. In a normal microcontroller, the bits and low level coding of the RF protocol are done by software. This is a very time consuming task for the CPU since the processor has to handle each bit as it is received or as it has to be transmitted. This block simplifies the handling of the low level data for wireless data transmission systems in a similar way to a UART interface for wired transmission systems.

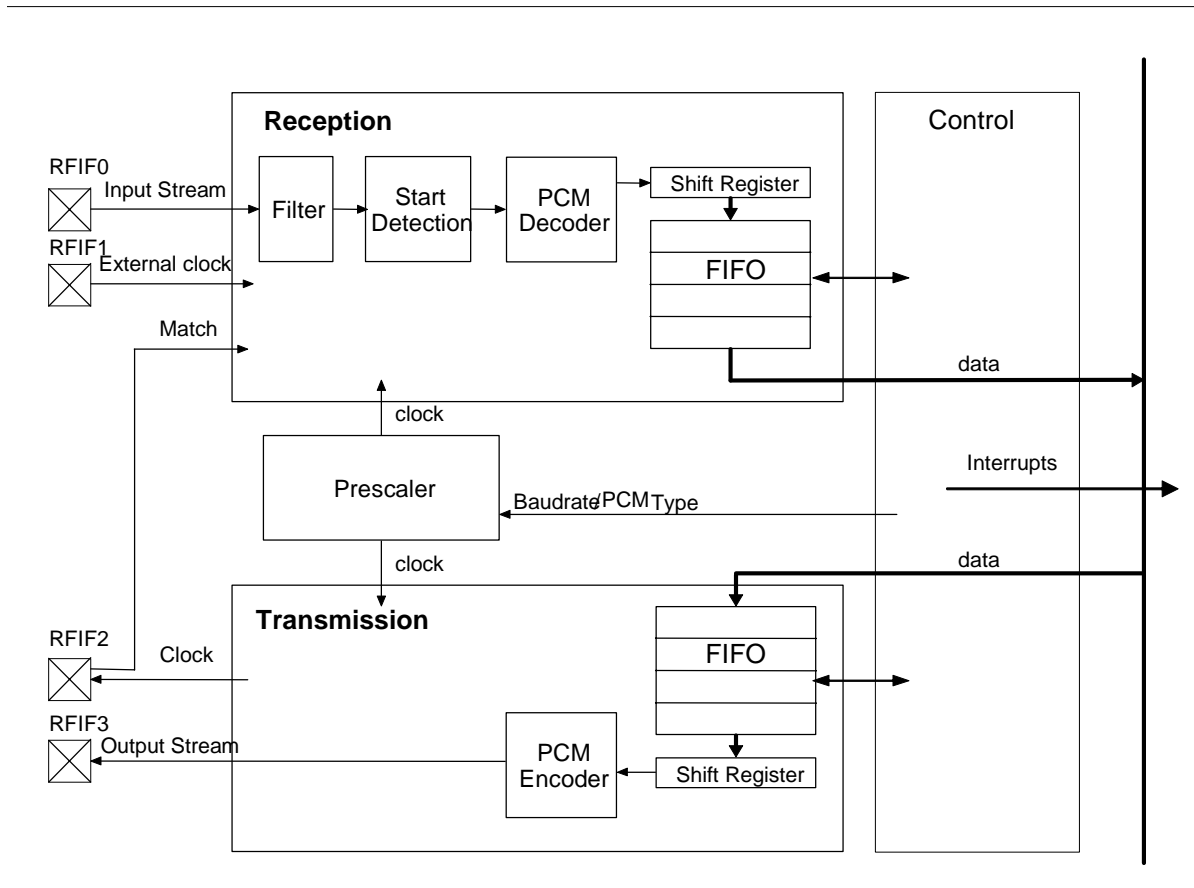


Figure 1 : Internal structure of the BitJockey™ interface

The **Reception** block consists of :

- The “Filter”. The input data is filtered according to the selected baud rate. The block extracts the bit synchronization clock. When an external bit synchronization clock is used, the filter is bypassed.
- The “Start Detection” block. This block detects a programmable start sequence and synchronizes on the message start. The interface ignores incoming messages until the start sequence is detected. This block can be bypassed or an external message synchronization signal can be used.
- The “PCM Decoder” which decodes the 3 main types of PCM waveforms (i.e. NRZ, Bi-Phase, Miller). This block can be bypassed.
- The “Shift Register” and the FIFO, which do the serial to parallel conversion and memorize the input data.

The **Transmission** block consists of:

- The FIFO and the “Shift Register”, which store the data and carry out the parallel to serial conversion.
- The “Encoder”, which encodes the serial data into the chosen PCM type. This block can be bypassed.

The **Prescaler** generates the clock signals needed for the reception and transmission according to the selected baud rate and PCM code type.

This RF interface can be used with Semtech's or third party RF transceiver circuits. Below is the figure illustrating the connections between the BitJockey™ and a transceiver with a Pattern output and no Clock input.

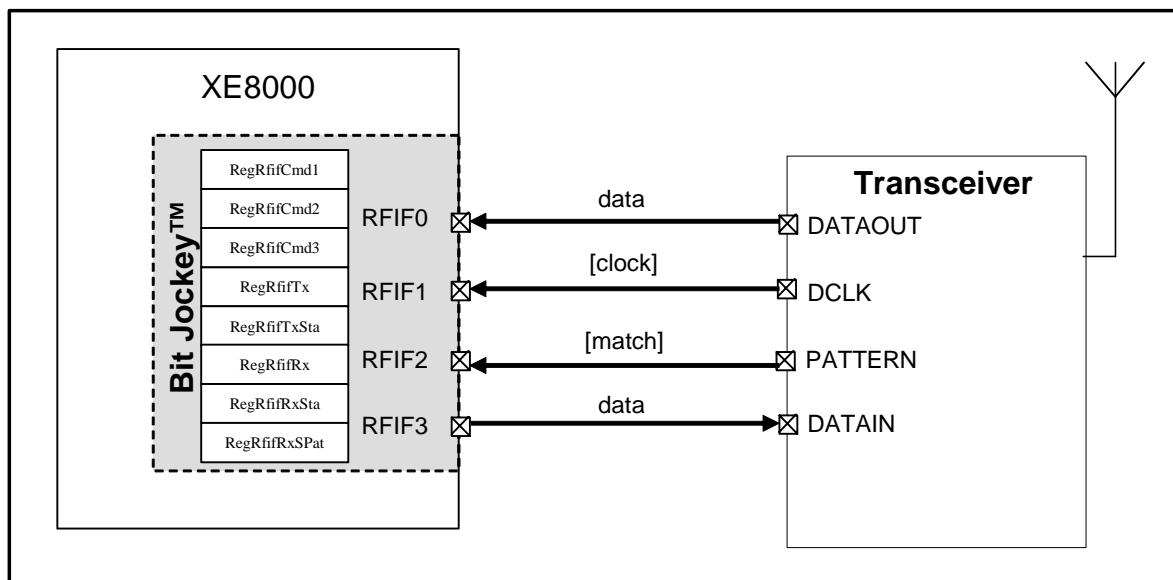


Figure 2: Bit Jockey™ <-> transceiver connections

For a complete description of the interface and operation of the BitJockey™ with the XEMICS SEMTECH XE12xx transceivers, please refer to section 7.2 of [2].

2. GENERAL GOAL OF THE DOCUMENT

This technical note's purpose is to provide a basic driver implementation example of the XE8000 BitJockey™ RF interface. For in-depth information please read the BitJockey™ chapter of the [1].

For this technical note we will take the hardware assumption described in the figure above but please note that depending on the transceiver or application, the DCLK and PATTERN pins may not be available. These two hardware connections are optional and if one still want to have their functionalities, the BitJockey™ interface can be configured to process them (clock recovery and pattern recognition).

3. GLOBAL DESCRIPTION

As illustrated in figure below, this driver simply implements a data transfer between the XE88LC06A RAM and the external transceiver via the BitJockey™ internal peripheral.

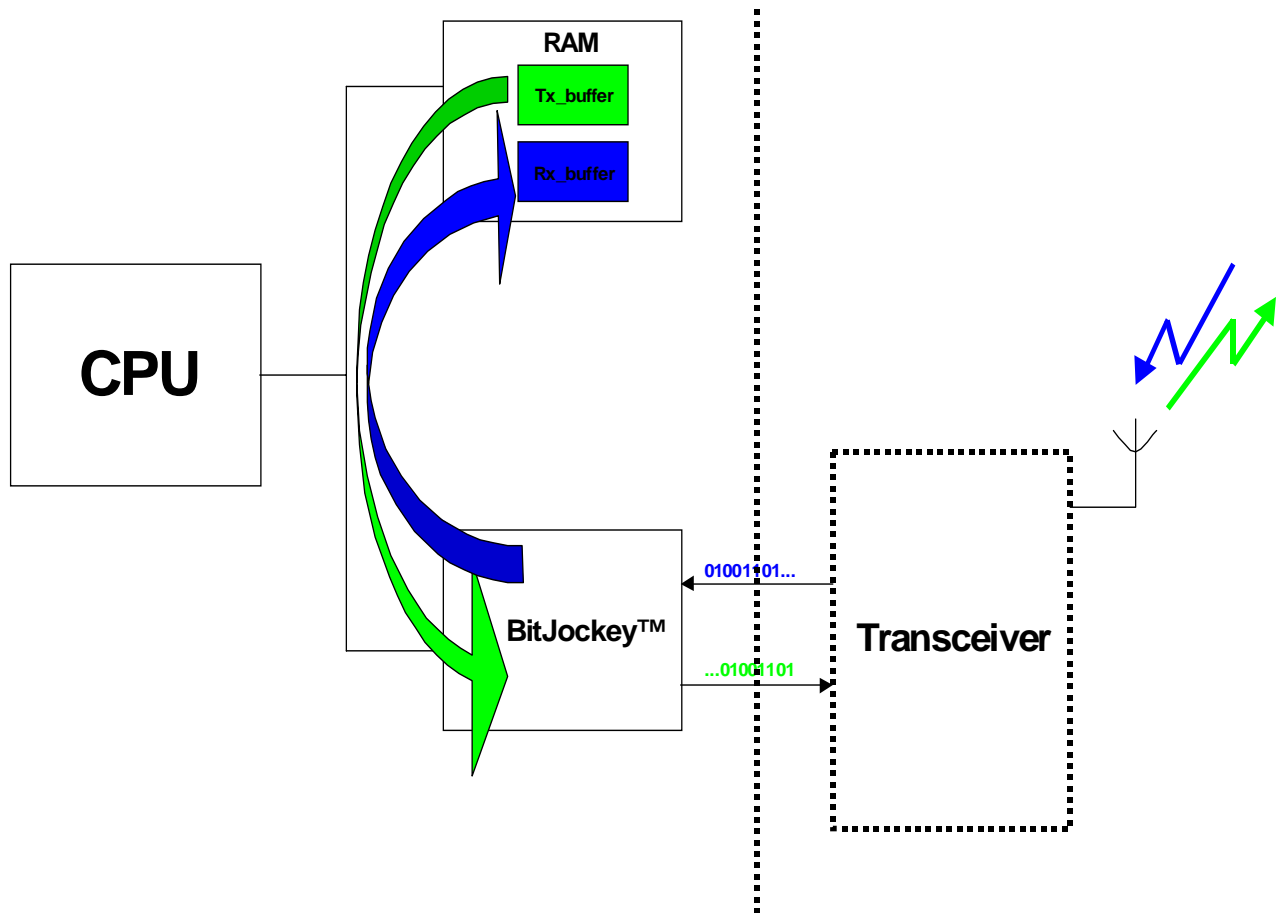


Figure 3: Driver's function

A typical application is a packetized RF link. In transmission mode the content of the transmission buffer starting at the Tx_buffer address in data memory is copied bit by bit to pin RFIF3 of the interface at the configured RF data rate. In reception mode the incoming packet is detected via a pattern ("Match" sequence) and the immediately following incoming bits on pin RFIF0 are stored in the reception buffer starting at the Rx_buffer address in data memory.

This driver can be used in a unidirectional or half duplex system. Depending on the application, the developer can set the interface permanently in transmission mode by calling `Asm_Initiate_Tx`, or solely in reception mode by calling `Asm_Initiate_Rx` or implement a half duplex RF link by calling one of the two functions, then the other, switching on a regular basis from one to the other.

4. DRIVER CONTENT

This driver is composed of two XE8000 assembler files:

Initiate_RF.s

This file implements two functions: `Asm_Initiate_Tx` and `Asm_Initiate_Rx`.

RF_Irq_handler.s

This file implements two functions: `Handle_Irq_Rfif_Tx` and `Handle_Irq_Rfif_Rx`

5. FUNCTIONS DESCRIPTION

5.1 Asm_Initiate_Tx

This function sets a transmission phase from the BitJockey™ interface to the external RF transmitter.

When called, this routine enables the Tx interrupt line to CPU, configures the different registers (`RegRfifCmd1`, `RegRfifCmd2` and `RegRfifCmd3`) and starts sending the first four consecutive bytes located at address "Tx_buffer". The position of the next bytes to be put into FIFO is saved in the RF pointer `Current_RF_Buffer`.

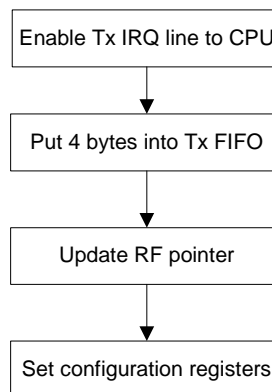


Figure 4: `Asm_Initiate_Tx` diagram flow

Note that the registers' default configuration implemented in this function is the following:

`RegRfifCmd1` = %00000010 Baud rate selection division factors set to 1 for Coarse and to 3 for Fine
`RegRfifCmd2` = %00000000 Bypass PCM encoder (implies NRZ-level coding) + No clock output
`RegRfifCmd3` = %00000001 Starts transmission

5.2 Asm_Initiate_Rx

This function sets a reception phase from the external RF receiver to the BitJockey™ interface.

When called, this routine enables the Rx interrupt line to CPU, configures the different registers (RegRfifCmd1, RegRfifCmd2 and RegRfifCmd3) and starts pattern detection. The received data following the pattern will be stored at memory address "Rx_buffer".

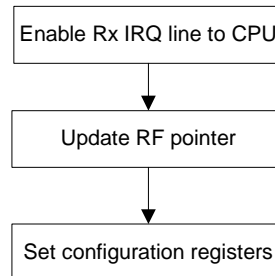


Figure 5: Asm_Initiate_Rx diagram flow

Note that the registers' default configuration implemented in this function is the following:

RegRfifCmd1 = %00000010 Baud rate selection division factors set to 1 for Coarse and to 3 for Fine
 RegRfifCmd2 = %10010000 External start detection mode + Clock input
 RegRfifCmd3 = %00100010 Sets "pattern detected" IRQ source and starts reception

5.3 Handle_Irq_Rfif_Tx

This function is automatically called by the crt0.s file when the BitJockey™'s Tx interrupt is raised, indicating that the transmission FIFO is empty.

As illustrated in the figure below, when called by crt0.s, the Handle_Irq_Rfif_Tx interrupt puts in the transmission FIFO the 4 four consecutive bytes located at address Current_RF_buffer.

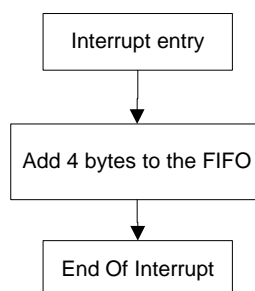


Figure 6: Transmission IRQ diagram flow

5.4 Handle_Irq_Rfif_Rx

This function is automatically called by the crt0.s file when the BitJockey™'s Rx interrupt is raised, independently of the source of the interrupt ("pattern detection" or "FIFO full"), indicating that the reception FIFO is full.

When a reception phase is started by Asm_Initiate_Rx, the interrupt source is set to "pattern detection" to start filling the FIFO only when a correct received data flow is detected. The first time the interruption is raised, the Handle_Irq_Rfif_Rx function will switch the Rx interrupt source from "Pattern detected" to "FIFO full" since received data are now being pushed into FIFO.

The following times Rx interrupt is raised, indicating that reception FIFO is full, the Handle_Irq_Rfif_Rx function will get 4 bytes from FIFO, store them in the Rx buffer at a memory address appointed by Current_RF_Buffer and update the RF pointer for the next FIFO full interrupt.

This mechanism is illustrated below:

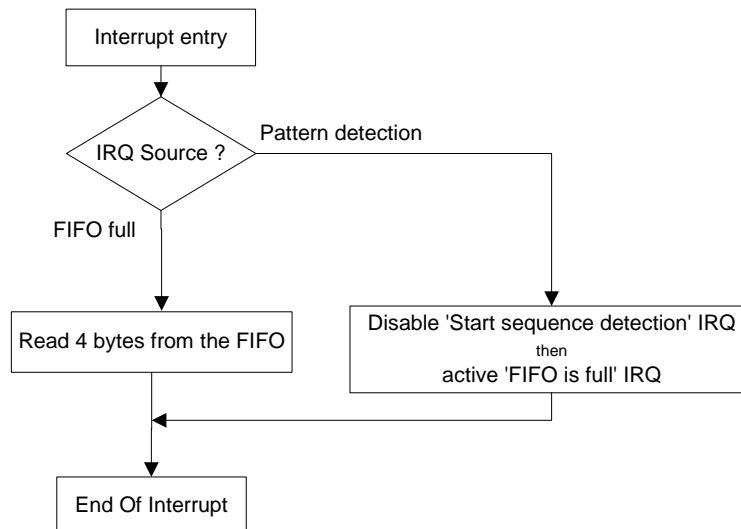


Figure 7: Reception IRQ diagram flow

6. HOW TO USE THE BASIC DRIVER

Define the configuration registers values for your system using the uC datasheet and replace the default values set in Asm_Initiate_Tx, Asm_Initiate_Rx and/or Handle_Irq_Rfif_Rx. Those modifiable values are in bold characters in the code.

Define somewhere in your code "Tx_buffer" or/and "Rx_buffer" with their respective size depending on your application.

The target specific crt0.s file must be used for compilation. (XE88LC06A, XE88LC07A, XE1283, ...)

The basic driver is now operational and can be used by calling one of initialisation functions, Asm_Initiate_Tx or Asm_Initiate_Rx. Note that Handle_Irq_Rfif_Rx and Handle_Irq_Rfif_Tx are automatically activated and called by crt0.s.

Since the code provided is a basic driver it can be easily customized by the developer to add other features like:

- ✓ A counter, initialised in `Asm_initiate_Rx/Tx` and decremented in `Handle_Irq_Rfif_Rx/Tx`, to count the number of received/sent bytes and identify the end of the frames if their size is constant.
- ✓ A Watch Dog, reset at every pattern detection in `Handle_Irq_Rfif_Rx`.
- ✓ The initialisation routines of the target transceiver (transmitter mode/reception mode), inserted at the very beginning of `Asm_initiate_Rx` and `Asm_initiate_Tx`.

7. REFERENCE

[1] XE88LC06A and XE88LC07A datasheet:

http://www.xemics.com/docs/xe8000/xe88lc06a_7_datasheet.pdf

[2] TN8000.18. XE8000 driving XE1200 transceivers. Standard API definitions.

http://www.xemics.com/docs/xe8000/tn8000_18_1200_api.pdf

8. INITIATE_RF.S

```
#####
##
## Initiate_RF.s: Transmission and reception initialisation routines ##
##
#####

.altregsyn 0

.global Asm_Initiate_Tx      ; Functions declared in this file
.global Asm_Initiate_Rx    ;

.global Tx_Buffer           ;
.global Rx_Buffer           ; Global variables used in this file

.global Current_RF_Buffer   ;
.global RF_counter          ; Global variables declared in this file

.section .page0_data

Current_RF_Buffer:         ; Pointer of the next received or transmitted RF byte
    .space 2                ;

.text

Asm_Initiate_Tx:

    move    a, RegIrqEnHig    ;
    or      a, #0x20          ; Enable Tx interrupt line to CPU
    move    RegIrqEnHig, a    ;

    move    i0h, #HIWORD(Tx_Buffer)    ; Load start address of Tx data
    move    i0l, #LOWORD(Tx_Buffer)    ;

    move    a, (i0)+          ;
    move    RegRfifTx, a     ;
    move    a, (i0)+          ;
    move    RegRfifTx, a     ; Send 4 bytes to FIFO
    move    a, (i0)+          ;
    move    RegRfifTx, a     ;
    move    a, (i0)+          ;
    move    RegRfifTx, a     ;

    move    Current_RF_Buffer, i0h    ; Update RF pointer
```

```

move    Current_RF_Buffer+1, i0l    ;

move    a, #0x02                    ; 00000010    Set baud rate
move    RegRfifCmd1, a ;

move    a, #0x00                    ; 00000000
move    RegRfifCmd2, a ;

move    a, #0x01                    ; 00000001    Set transmitter mode
move    RegRfifCmd3, a ;

jump ip

```

Asm_Initiate_Rx:

```

move    a, RegIrqEnHig    ;
or      a, #0x80          ; Enable Rx interrupt line to CPU
move    RegIrqEnHig, a    ;

move    i0h, #HIWORD(Rx_Buffer)    ;
move    i0l, #LOWORD(Rx_Buffer)    ;
                                ; Initialize the RF pointer
                                ; Received data will be stored in Rx_buffer

move    Current_RF_Buffer, i0h    ;
move    Current_RF_Buffer+1, i0l    ;

move    a, #0x02          ; 00000010    Set baud rate
move    RegRfifCmd1, a ;

move    a, #0x90          ; 10010000    Set external start detection mode + Clock input
move    RegRfifCmd2, a ;

move    a, #0x22          ; 00100010    Enable pattern interrupt + Set receiver mode
move    RegRfifCmd3, a ;

jump ip

```

9. RF_IRQ_HANDLER.S

```

#####
##
## RF_IRQ_handler.s: Handling functions of BitJockey™ interface interrupts ##
##                               ##
#####

.altregsyn 0

.global Current_RF_Buffer    ;; Global variable used in this file

.global Handle_Irq_Rfif_Rx
.global Handle_Irq_Rfif_Tx

.text

;*****;
;*                                           *;
;*           RF INTERFACE :           RX INTERRUPT HANDLING           *;
;*                                           *;
;*****;

Handle_Irq_Rfif_Rx:

move    RegIrqHig, #0x80 ; Clear IRQ flag

move    a, RegRfifCmd3    ;
tstb   a, #5              ;; if Pattern interrupt then jump to Pattern_detected
jzc    Pattern_detected ;

```

```

;-----;
;-          FIFO_full interrupt source          -;
;-----;

move    i0h,Current_RF_Buffer    ; Read Current_RF_Buffer position
move    i0l,Current_RF_Buffer+1  ;

move    a, RegRfifRx    ;
move    (i0)+, a        ;
move    a, RegRfifRx    ;
move    (i0)+, a        ;; Get 4 bytes from FIFO
move    a, RegRfifRx    ;
move    (i0)+, a        ;
move    a, RegRfifRx    ;
move    (i0)+, a        ;

move    Current_RF_Buffer, i0h    ; Save Current_RF_Buffer new position
move    Current_RF_Buffer+1, i0l  ;

jump    ip

;-----;
;-          Pattern_Detected interrupt source    ;
;-----;
Pattern_detected:

move    a, #0x82          ; 10000010  Enable FifoFull Interrupt source + Receiver
move    RegRfifCmd3, a   ;

jump    ip

;*****;
;*                                           *;
;*          RF INTERFACE :          TX INTERRUPT HANDLING          *;
;*                                           *;
;*****;

Handle_Irq_Rfif_Tx:

move    RegIrqHig, #0x20          ; Clear IRQ flag

move    i0h,Current_RF_Buffer    ; Read Current_RF_Buffer position
move    i0l,Current_RF_Buffer+1  ;

move    a, (i0)+    ;
move    RegRfifTx, a ;
move    a, (i0)+    ;
move    RegRfifTx, a ; Send 4 bytes to FIFO
move    a, (i0)+    ;
move    RegRfifTx, a ;
move    a, (i0)+    ;
move    RegRfifTx, a ;

move    Current_RF_Buffer, i0h    ; Save Current_RF_Buffer new position
move    Current_RF_Buffer+1, i0l  ;

jump    ip

```

© Semtech 2005

All rights reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey nor imply any license under patent or other industrial or intellectual property rights. Semtech. assumes no responsibility or liability whatsoever for any failure or unexpected operation resulting from misuse, neglect improper installation, repair or improper handling or unusual physical or electrical stress including, but not limited to, exposure to parameters beyond the specified maximum ratings or operation outside the specified range.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.

Contact Information

Taiwan	Tel: 886-2-2748-3380 Fax: 886-2-2748-3390	Switzerland	Tel: 41-32-729-4000 Fax: 41-32-729-4001
Korea	Tel: 82-2-527-4377 Fax: 82-2-527-4376	United Kingdom	Tel: 44-1794-527-600 Fax: 44-1794-527-601
Shanghai	Tel: 86-21-6391-0830 Fax: 86-21-6391-0831	France	Tel: 33-(0)169-28-22-00 Fax: 33-(0)169-28-12-98
Japan	Tel: 81-3-6408-0950 Fax: 81-3-6408-0951	Germany	Tel: 49-(0)8161-140-123 Fax: 49-(0)8161-140-124

Semtech International AG is a wholly-owned subsidiary of Semtech Corporation, which has its headquarters in the U.S.A