Application Report
SLAA121 – March 2001

Implementing a Bidirectional, Half-Duplex FSK RF Link With TRF6900 and MSP430

Peter Spevak *MSLP – MSP430*

ABSTRACT

The advantage of radio frequency (RF) over infrared (IR) links for communication and data transfer is that with RF a successful communication can be set up even when the participants are out of sight. This application report uses the MSP-EVKTRF6900 (Demo 1.16) evaluation kit (EVK) to show how easy it is to implement an RF link, and it shows how the setup is made code-efficient by using the MSP430 microcontroller. Implementation is discussed principally from the standpoint of the microcontroller; however, RF-related issues are also treated. The code and logic flowcharts for the TRF6900 Demo 1.16 RF link are included.

Contents

Figures

Tables

1 Introduction

TI's new single-chip RF transceiver, TRF6900, is intended for linear (FM) and digital (FSK) modulated applications in the new 868-MHz European and 915-MHz North American ISM bands for wireless communication data transfer. The TRF6900 is available in a 48-lead TQFP package and is designed as a fully-functional multichannel FM transceiver. The single chip operates down to 2.2 V, and is specifically designed for low power consumption. The synthesizer has typical channel spacing of approximately 230 Hz to accommodate both narrow- and wide-band applications. Because of the high frequency resolution of the direct digital synthesizer (DDS), the DDS can be used to fine tune the TX/RX frequency and allows the use of inexpensive reference crystals.

Two fully-programmable operation modes, Mode0 and Mode1, allow extremely fast switching between two preprogrammed settings, e.g., receive/transmit (RX/TX), TX_frequency_0/TX_frequency_1, and RX_frequency_0/RX_frequency_1 without having to reprogram the device. Each functional block of the transceiver can be enabled or disabled via the serial interface.

The TRF6900 needs only a few passive components to build up a complete working application on the RF side. The application needs a microcontroller to drive the transceiver via its serial interface—the MSP430 family of ultralow-power microcontrollers from TI fits the needs of such an RF link.

This report describes an example application that was set up on the MSP-EVKTRF6900. It gives all the information needed for adapting the RF design to specific needs, including the complete documentation of the MSP430 driven application and the source code with flowcharts.

2 Description of the TRF6900

There are two versions of the TRF6900 RF link, namely, Demo 1.11 and Demo 1.16.

2.1 Demo 1.11

This section provides a short description of the RF link used to implement the TRF6900 Demo 1.11. This was the predecessor of TRF6900 Demo 1.16 and was the first version of the RF link to be implemented on the MSP-EVKTRF6900. It was a very simple bidirectional RF link, without any acknowledge capability. Figure 1 shows the phases during the transmission of a data package using this RF link.

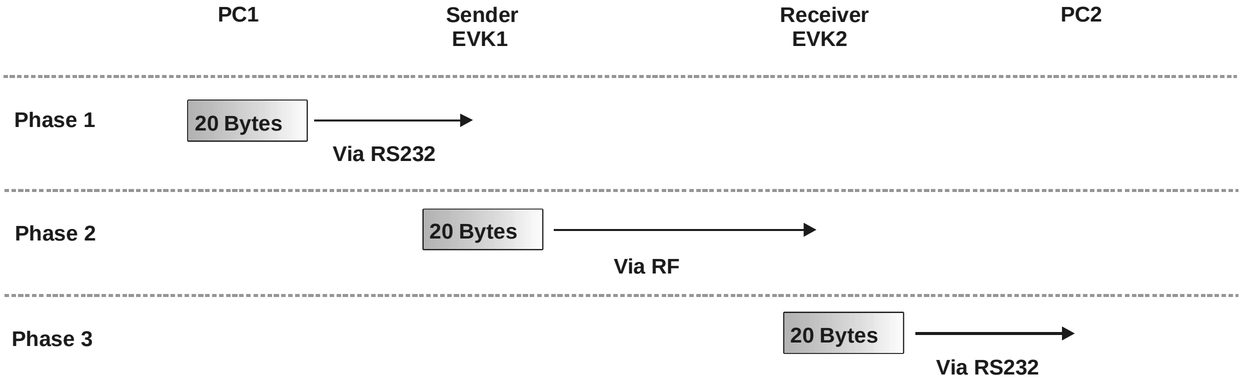


Figure 1. Phases of Data Transmission Using TRF6900 Demo 1.11

Figure 1 shows that the transmitting EVK is not acknowledged by the receiving EVK. Both EVKs, i.e., both ends of the RF link, are able to send and receive data. The EVKs run in the system mode and, by default, both are in the receive mode. As soon as an EVK receives a data package from the RS232 port, it switches to transmit mode and broadcasts the data package via RF. Any other EVK, provided it is in receive mode and is within range of the transmitting EVK, receives the data without sending an acknowledge. This is why such a link can be called a broadcasting system.

2.2 Demo 1.16

Figure 2 shows the different phases of communication flow implemented in the TRF6900 Demo 1.16.

The main difference from Version 1.11 is acknowledgement of the received data. Referring to Figure 2, in phase 2 the receiving EVK calculates a checksum of the whole data package. In Phase 3 the EVK transmits the data package (including the checksum) via RF. In Phase 4, the checksum of the received data package is recalculated by the receiving EVK and compared with the checksum received from the sending EVK. Phases 5 and later (marked with an asterisk in Figure 2) occur only when there is equality of the checksums. In that case, the receiving EVK sends an acknowledge to the sending EVK, and transmits the received data package to the connected PC.

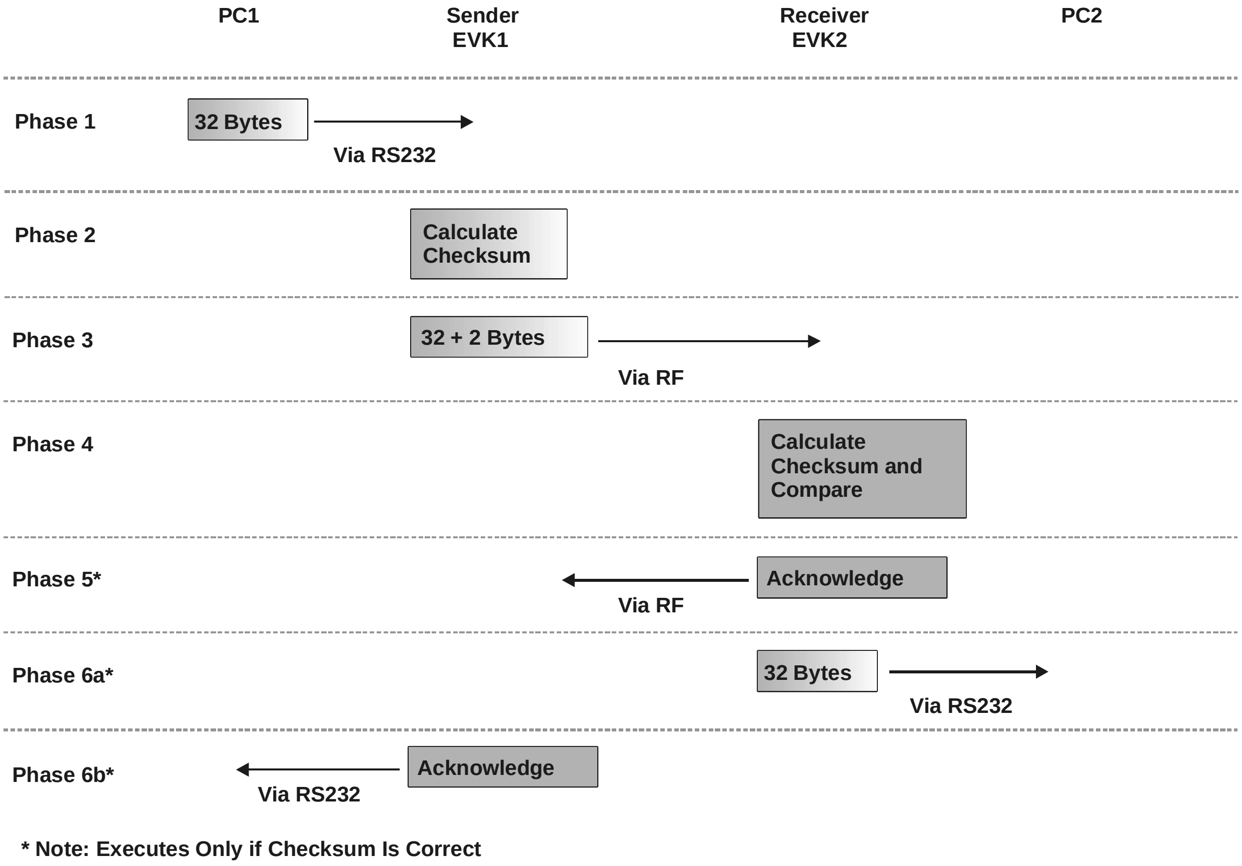


Figure 2. Phases of Data Transmission for TRF6900 Demo 1.16

3 Specification of RF and RS232 Protocols

3.1 RS232 Protocol

The data format used for the RS232 communication is as follows:

Data rate:	19.2 kbps
Resulting bit length:	52.08 µs
Data package:	32 bytes, a 1 start bit, 8 data bits and 1 stop bit

The signal on the TX-line of the MSP430 microcontroller (RS232 line) is shown in Figure 3. As the RS232 signal is inverted, the TX-line is low by default (microcontroller pin is high). The start bit is a high pulse, 52.08 µs long. Then the first data bit is sent, beginning with the LSB. After the package of 8 data bits, the stop bit arrives, generating a low pulse on the RS232 line and a high pulse on the MSP430 pin.

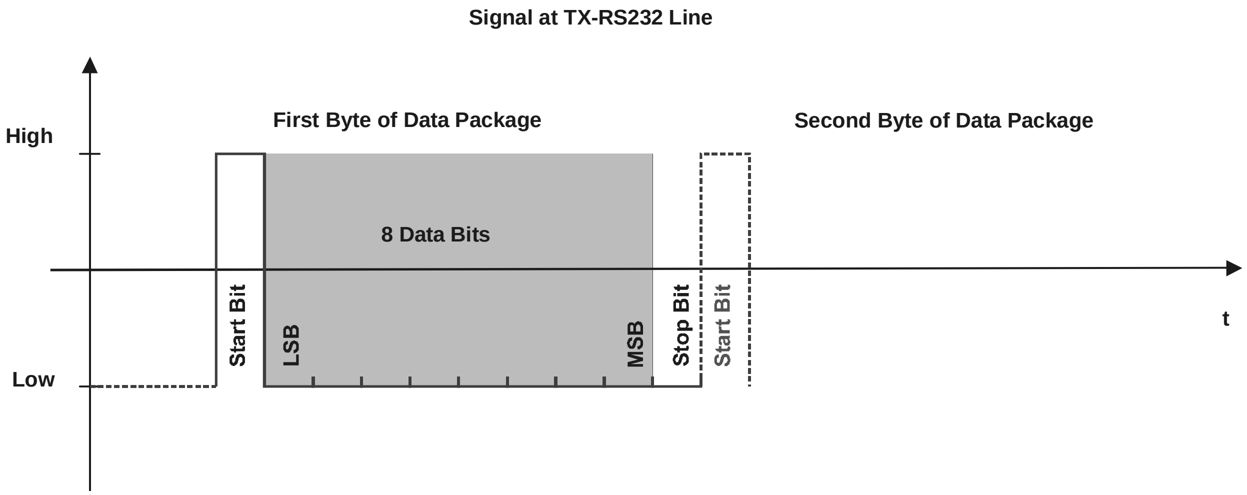


Figure 3. RS232 Protocol

3.2 RF protocol

Since the release of Revision A of MSP-EVKTRF6900, the code for the RF link has been improved, and the format of the RF protocol has changed slightly. Unless otherwise stated, this document always refers to the October 2000 version, released October 2000. The numerical values in the brackets refer to the format used in Revision A. The data format used for the RF communication is as follows:

Data rate:	38.4 kbps
Resulting bit length:	26.04 µs
Data package:	Training sequence 1ms (5ms), one Start Bit, 32 + 2 Byte (32 Bytes)
Acknowledge:	Training sequence 1ms, Start Bit, 2 Bytes - Checksum of the received data, (the Revision A firmware does not implement any Acknowledge)
Coding:	NRZ (Non return-to-zero)

Figure 4 shows the RF protocol of the implemented RF link. The whole sequence of the RF protocol consists of three parts: the training sequence, the start bit, and the data package that includes 32 bytes of data. These three parts are discussed in the following sections.

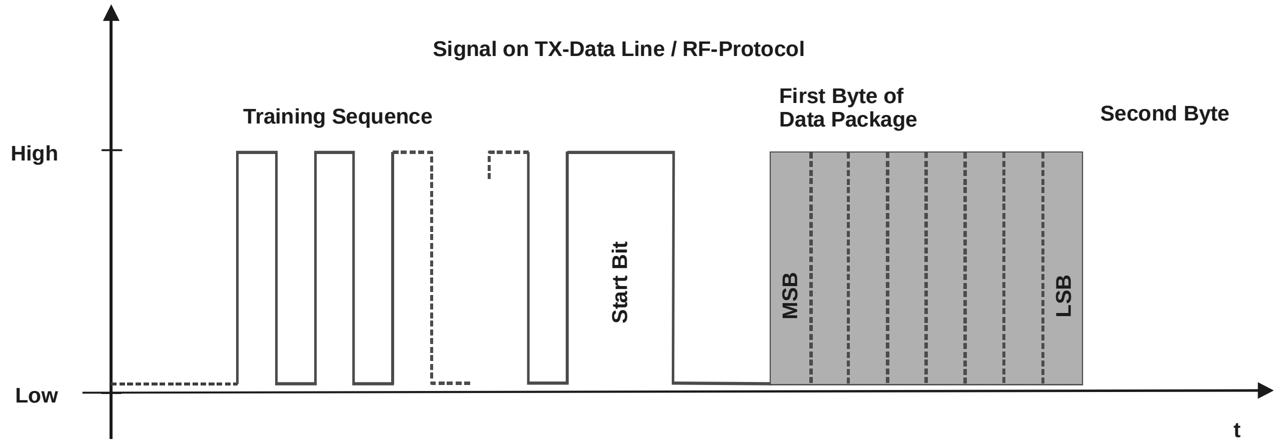


Figure 4. RF protocol

3.2.1 Training Sequence

Besides the ability to receive return-to-zero coded signals, the TRF6900 transceiver features the availability to receive of non-return-to-zero (NRZ) coded signals. This doubles the maximum achievable data transmission rate, compared with biphase or Manchester coding. Refer to section 3.2.3 for a detailed explanation.

This feature is supported by the data slicer, the external sample-and-hold capacitor, and the automatic frequency control (AFC) loop. The AFC loop shown in Figure 5 charges the sample-and-hold capacitor to the dc value of the received signal. The capacitor is charged up during reception in *learn-mode*. This happens during reception of the training sequence. The training sequence is a square wave signal of equal high and low pulses. The voltage across the sample-and-hold capacitor is charged up to the dc value of the received and demodulated signal. After a long enough period to achieve good accuracy of this reference voltage value, the TRF6900 is switched from reception in *learn-mode*, to the reception in *hold-mode*. In the hold mode, the voltage across the sample-and-hold capacitor is maintained. The maximum hold time of the TRF6900 is determined by the values of the internal and external parasitic resistances, (which determine the discharge current), and the value of the sample and hold capacitor. Typically, the maximum hold time is several seconds. This makes the reception of a signal with many successive ones or zeros possible.

In summary, the training sequence has two purposes: The first is to enable the receiver to adapt to the transmitted signal, charge up the sample-and-hold capacitor to the dc value of the signal, and thereby enable reception of NRZ-coded signals. The second is to enable the receiver to distinguish between noise (or invalid signals in the scanned frequency band) and valid data.

Achieving these purposes allows the minimum length of the training sequence to be calculated. The following factors must be considered: run-in-time of the receiver, e.g., from standby to receive modes (typically 600 µs), the time during which the microcontroller on the receiver side needs to recognize the training sequence, and some overhead to minimize the number of training sequences that are unrecognized because of timing problems. The training sequence of the Revision A of MSP-EVKTRF6900 was about 4 ms duration. The code for the MSP430P112 and MSP430F1121, written for the RF link of the TRF6900 Demo 1.16, has been gradually improved, so that the currently available code uses a training sequence of less than 1 ms duration, resulting in even better performance.

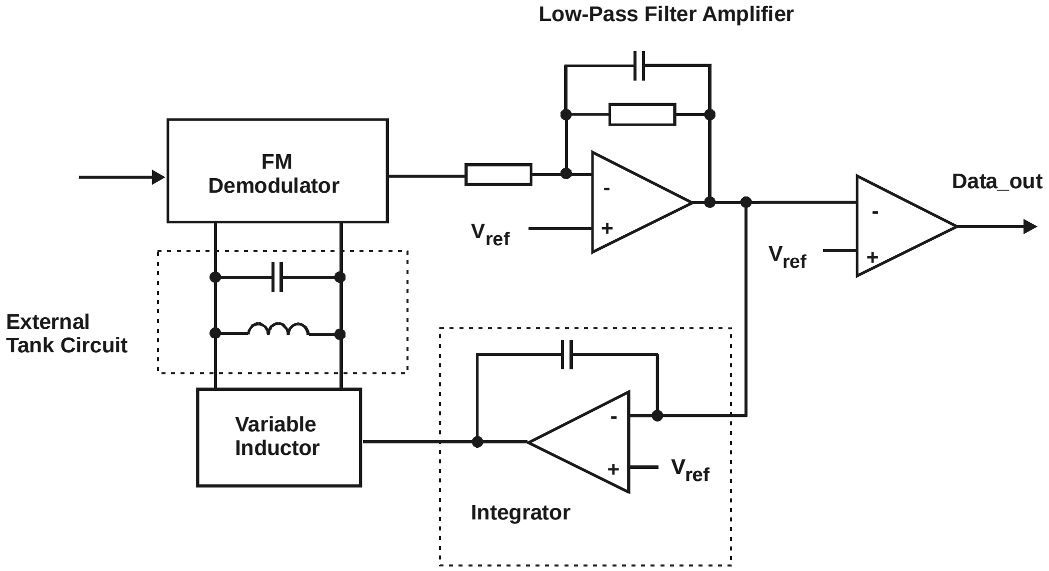


Figure 5. Automatic Frequency Control Loop

Figure 6 shows the signal of the training sequence used for the implemented RF link on the MSP-EVKTRF6900. As mentioned before, the training sequence on the Revision B EVK is below 1ms long. During this time the receiver has to detect and recognize the signal. The AFC circuit has to adapt to the signal of the sender. After recognition of the training sequence, the TRF6900 has to be switched from reception in *learn-mode* to reception in *hold-mode*, to enable the TRF6900 to receive NRZ code.

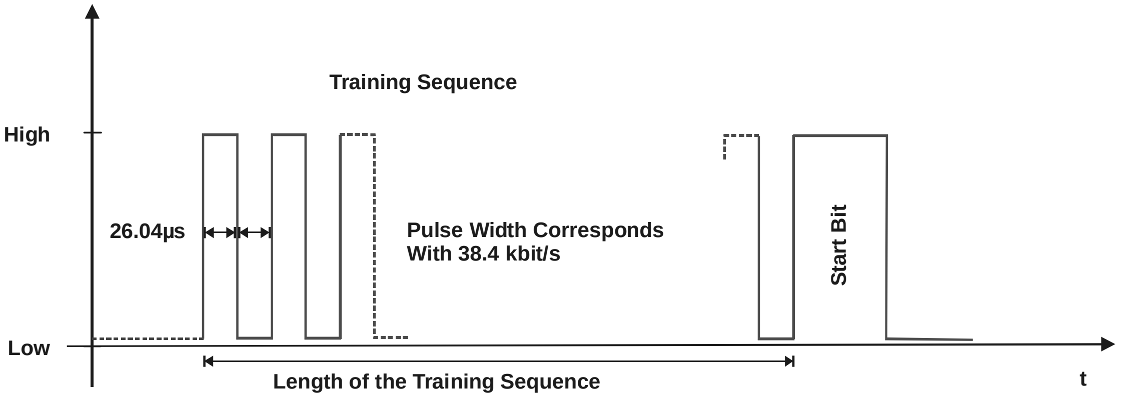

Training Sequence
High
Low
26.04µs
Pulse Width Corresponds With 38.4 kbit/s
Start Bit
Length of the Training Sequence
t

Figure 6. Training Sequence

3.2.2 Start Bit

The start bit is shown Figures 4 and 6. The purpose of the start bit is to enable the reception of the data package by marking the beginning of the actual data package, and to synchronize the receiver. The length of the start bit can be within a wide range, but it must enable a distinction to be made easily between the pulses of the training sequence and the pulse of the start bit. In the case of the EVK, the pulse width was set to three times the pulse length of the training sequence, giving a start bit of 78.12 µs duration.

3.2.3 Data Package

The data package is that part of the RF protocol that contains the actual data to be transmitted. The Revision-A EVK transmits only 20-byte data packages. The code listings and associated flowcharts can be found in Appendices A to R.

The October 2000 code release implements a data link for transmitting 32 data bytes, as well as for transmitting two additional bytes containing the checksum for these 32 data bytes. For data transmission, the first two transmitted bytes are the checksum of the transmitted data package as calculated by the sender, followed by the 32 bytes of actual data. These 34 bytes in the data package are not separated by start or stop bits. Transmission of a single byte always begins with the most significant bit (MSB) of the transmitted byte.

Figure 7 shows the shape of the data package when sending 32 data bytes, and Figure 8 shows the shape when of sending the acknowledge for the received data package.

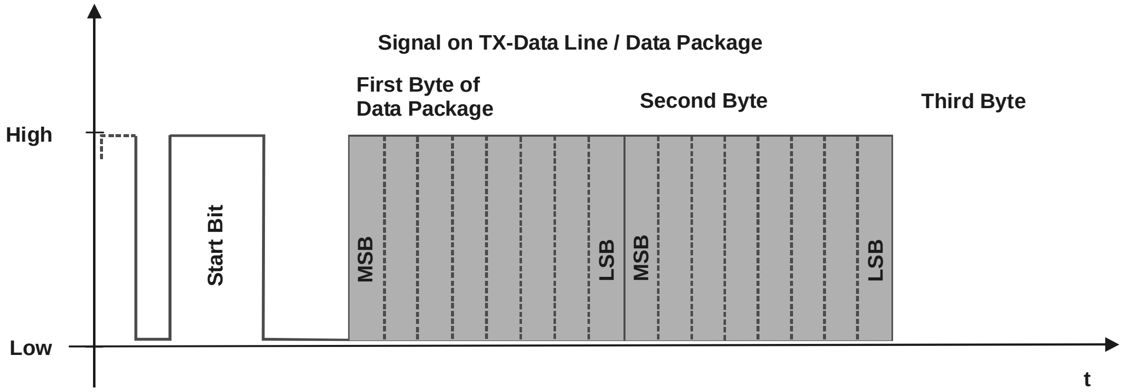

Signal on TX-Data Line / Data Package
First Byte of Data Package
Second Byte
Third Byte
High
Low
Start Bit
MSB
LSB
MSB
LSB
t

Figure 7. Data Package—Sending 32 Bytes of Data

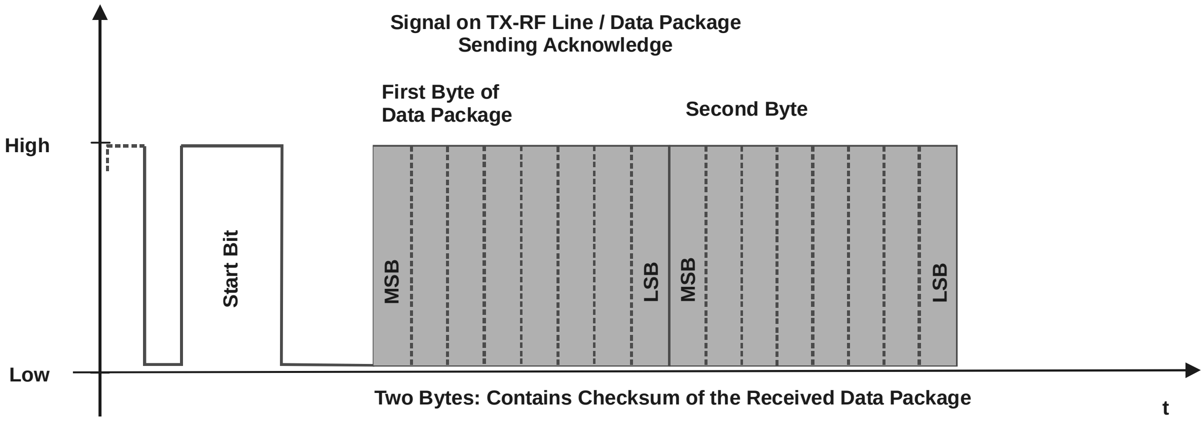

Signal on TX-RF Line / Data Package
Sending Acknowledge
First Byte of
Data Package
Second Byte
High
Low
Start Bit
MSB
LSB
MSB
LSB
Two Bytes: Contains Checksum of the Received Data Package
t

Figure 8. Data Package for Sending Acknowledge

4 Implementation of RF Reception

This section discusses how the RF signal is received, by describing (i) the TRF6900 settings that must be programmed by the microcontroller to enable reception, (ii) the code by means of flowcharts, (iii) detection and recognition of the training sequence, and (iv) actual data reception and the sending of the acknowledge. The firmware described here is the October 2000 release. The code and flowcharts of the Revision-B EVK are contained in Appendices A to R and can be compared with the improved version described in this section.

4.1 Programming the TRF6900

The TRF6900 features two pre-programmable modes that can be freely preprogrammed through the serial interface of the TRF6900 for transmission or reception. A block diagram of the serial interface is shown in Figure 9.

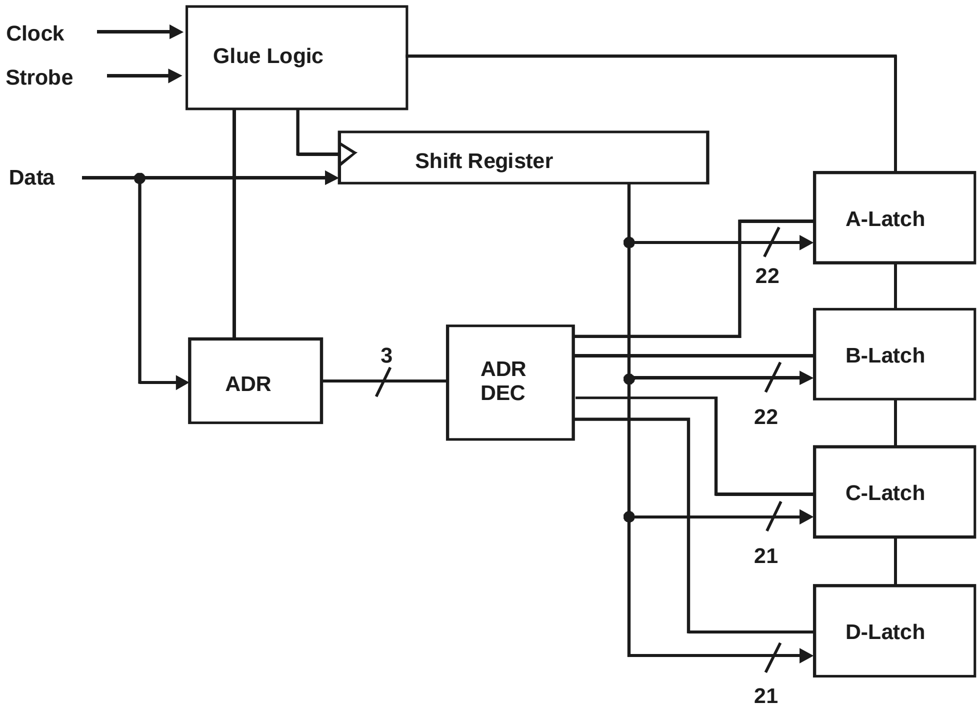


Figure 9. TRF6900—Serial Interface

The TRF6900 has four registers: A-Latch and D-Latch are for Mode 0 while B-latch and C-latch are for Mode 1. A-latch and B-latch are 22 bits wide and are used to program the settings of the direct digital synthesizer (DDS) in the respective modes. The two MSBs of the 24-bit wide shift register are used for addressing the latches.

The C-latch and D-latch contain the settings for control of the TRF6900, and the settings for all the modules of the TRF6900 must be done here. These two latches are 21 bits wide and, in this case, the three MSBs are used for addressing the latches.

The settings for these four latches are programmed via the unidirectional serial bus, consisting of DATA, CLOCK and STROBE signals. With every rising edge of the clock signal, the applied value at the data line is moved into the shift register. Program execution starts with the MSB. The content of the shift register is pushed into the addressed latches with the rising edge at the strobe line. At this instant, the DATA and CLOCK signals should be low. Figure 10 shows an oscilloscope trace of program execution on the EVK. For technical data on the timings during the programming, see *TRF6900 Single-Chip RF Transceiver*, Literature Number SLAS213d [1].

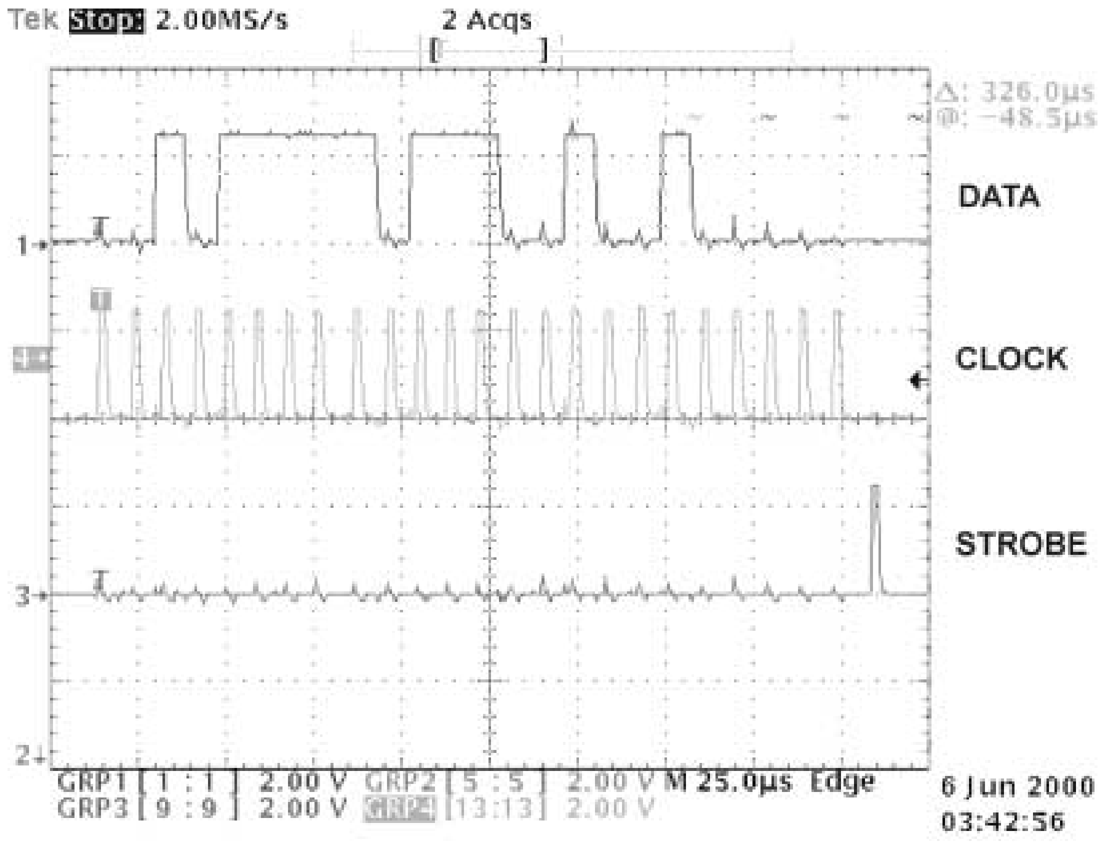


Figure 10. Programming the TRF6900 (Word A)—MSP-EVKTRF6900

There are five different settings that are used for programming the modes of the TRF6900 to implement the RF link with the EVK. The routines and settings are explained in following sections.

4.1.1 Frequency Settings for the Implemented RF Link

The first two routines are used for the two different frequency settings for the TRF6900. One is needed for the send-mode (Mode 1), and the other is for the receive-mode (Mode 0). As already mentioned, the RF link implemented on the EVK is set up at a center frequency of 869.83 MHz with frequency shift keying (FSK). FSK means that the information in the transmitted data is encoded by using two different frequencies. These two frequencies differ from each other by the selected deviation which, in the case of the EVK, is set to 60 kHz. This is demonstrated in Figure 11.

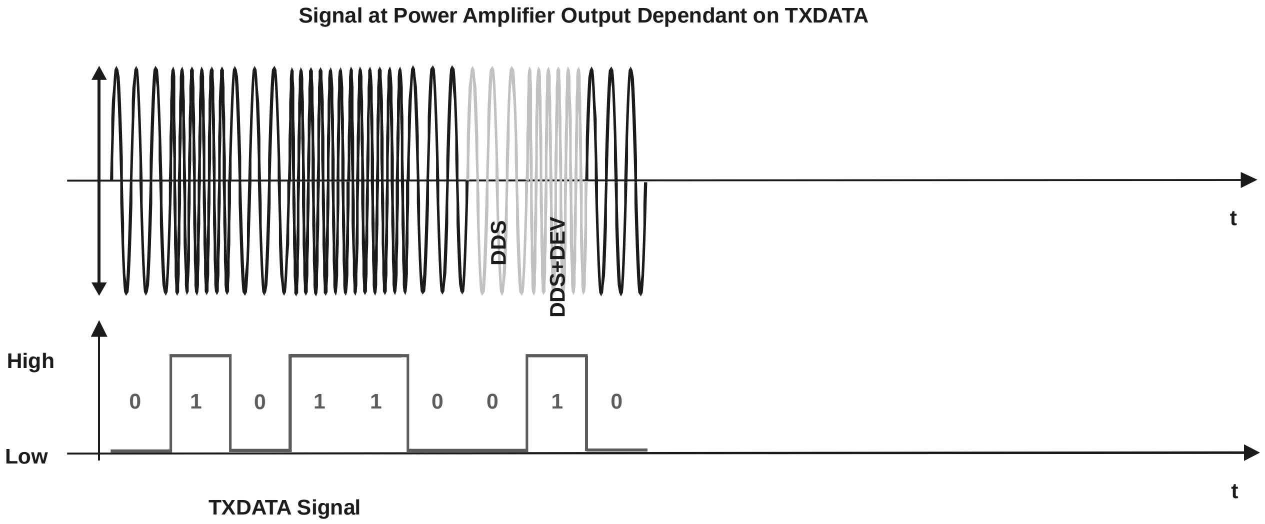


Figure 11. Output Signal of Power Amplifier—Dependant on TXDATA Signal

The frequency of the output signal from the power amplifier (PA) toggles between the two values, depending on the TXDATA signal. As long as TXDATA is low, the PA output frequency corresponds with the DDS settings for the send mode (Mode 1 / DDS 1 / 869.83 MHz). As soon as logic 1 is applied to the TXDATA input of the TRF6900, the frequency rises to the value equal to DDS 1 plus the value programmed into the deviation bits in the D-word. For the EVK, this corresponds to a value of 869.89 MHz (869.83 MHz + 60 kHz).

The receiver part of the EVK works with an intermediate frequency (IF) of 10.7 MHz. There are several reasons why a receiver working with a fixed IF has advantages. First, the filters that must separate the signal from the noise outside the frequency band used by the transmitted signal can themselves be adjusted to a certain frequency band. This strategy is much easier and less expensive to implement than adaptive filters. Second, because of the fixed IF, the filtering characteristic of the receiver remains constant over the whole band used by the RF link. This frequency band is determined by the limits of the voltage-controlled oscillator (VCO) and the phase-locked loop (PLL). Another reason to use such a receiver is that the circuitry of the receiver has to deal with a much lower frequency, which helps to reduce losses. To transform the received signal information to the IF, the received frequency is mixed with a frequency that differs by the value of the IF. The result is a signal at the IF frequency. For that reason, the DDS 0 frequency (Mode 0) is set to 859.13 MHz. This frequency is lower than the frequency of the sender by 10.7 MHz. When these two frequencies are mixed together, the result is a 10.7 MHz signal containing the information in the transmitted signal in the form of 60-kHz frequency modulation.

In principle the received signal can be mixed down to any frequency, but because of the availability and cost of filter components, it is useful to use an IF that is often used for other applications, too. That is why an IF of 10.7 MHz has been chosen for the EVK—it is the same IF as is used in ultrashort-wave radio systems.

The formula for the calculation of the bit settings for the two DDS registers is:

$$f_{(out)} = N \cdot \frac{f_{(\mathrm{ref})}[DDS\_x]}{2^{24}}$$

where *N* is the multiplication factor of the PLL, *$f_{(ref)}$* is the frequency of the reference oscillator and *[DDS_x]* is the content of the active DDS register. The value calculated by this formula may differ from the measured value because the tolerances of the external components, affect the precision of the reference oscillator; therefore, the settings for the DDS and deviation registers should be checked by measurement.

As already mentioned above, in addition to the two DDS registers that define the base frequencies of the VCO for Mode 0 and Mode 1, there is a third register that controls the frequency deviation when FSK is selected. This is the D-word register (Bit [20:13]). The formula for its setting is:

$$\Delta f_{2-FSK} = N \cdot \frac{f_{(ref)}\,DEV}{2^{22}}$$

The frequency at the output of the power amplifier is:

$$f_{(out:TXDATA=Low)} = N \cdot \frac{f_{(\mathrm{ref})}[DDS\_x]}{2^{24}}$$

$$f_{(out:TXDATA\ =High)} = N \cdot \frac{f_{(\mathrm{ref})}([DDS\_x] + 4\,DEV)}{2^{24}}$$

4.1.2 Control Register Settings (C- and D-Word Registers)

Besides the two DDS frequency registers, there are two control registers, namely the C-word and D-word registers. The control bit in each register is valid only when the correct mode is selected for it: Mode 1, C-word; Mode 0, D-word. However, as shown in Figure 12, there are also control bits that are valid independent from the selected mode.

For the RF link implemented on the MSP-EVKTRF6900, two different settings for the C-word and one setting for the D-word have been used. The two different settings for the C-word are for switching between reception-in-learn and reception-in-hold modes. Reception-in-hold mode enables the TRF6900 to receive a non-dc-free signal, and makes the reception and use of NRZ encoded signals possible. The control bit for this setting is implemented in the C-word, which requires this register to be reprogrammed to do the switching.

As explained later in this report, using NRZ encoded signals saves transmission time.

C-Word: Validity of Control Bits

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	X	X	X	X	X	X	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X
Address Bits			APLL			NPLL	MM	SLCTL			PLL	VCO	PA		SLC	LPF	SW	RSSI	DEM	IF	MIX	LNAM	

▓ Valid Independent of Selected Mode

░ Valid Only When Mode 1 Selected

D-Word: Validity of Control Bits

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Address Bits			DEV								PLL	VCO	PA		SLC	LPF	SW	RSSI	DEM	IF	MIX	LNAM	

▓ Valid Independent of Selected Mode

░ Valid Only When Mode 0 Selected

Figure 12. Validity of Control Bits in the C- and D-Word Registers

Figure 13 compares the transmission of the same bit sequence, one biphase-encoded, and in the other example as an NRZ-encoded baseband signal. In both cases the minimum pulse width is assumed the same. The minimum pulse width can be determined by the baseband, or let us say, by the maximum clock frequency of the microcontroller, as well as by the limitations of the RF portion of the implemented RF link. Of course talking about the maximum clock frequency of the used microcontroller used in the application does not really mean the minimum pulse width, the microcontroller is able to generate. Rather, it has to be considered that the minimum usable pulse width is determined by the ability of the microcontroller to detect and receive such a signal; this is certainly affected mainly by the speed of the microcontroller.

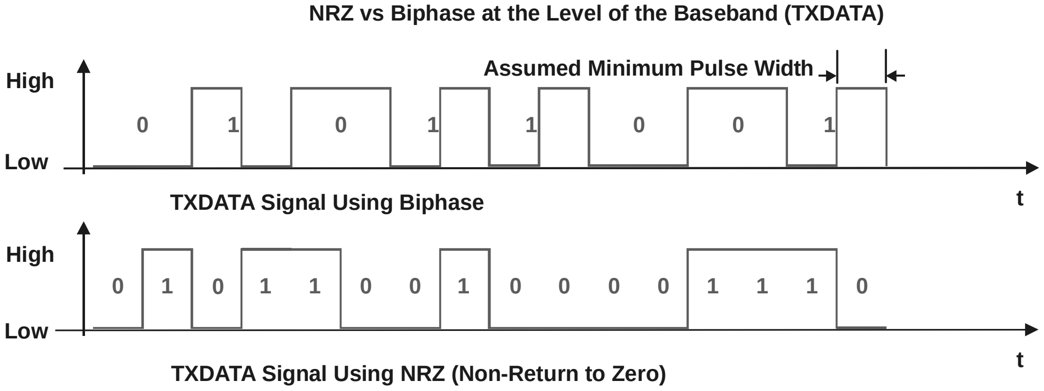
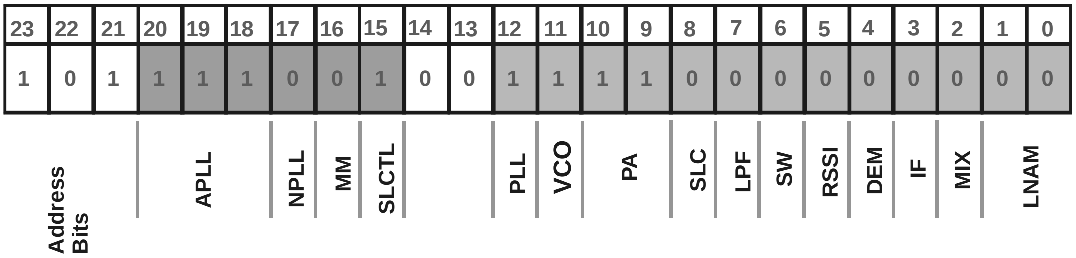

NRZ vs Biphase at the Level of the Baseband (TXDATA)
Assumed Minimum Pulse Width
High
Low
0 1 0 1 1 0 0 1
t
TXDATA Signal Using Biphase
High
Low
0 1 0 1 1 0 0 1 0 0 0 0 1 1 1 0
t
TXDATA Signal Using NRZ (Non-Return to Zero)

Figure 13. Comparison of NRZ and Biphase-Encoded Baseband Signals

The two different settings for the C-word register are set up by the *program_send_FSK* and *program_receive_FSK_hold* routines which send the settings for the TRF6900 to the universal programming routine *program_TRF6900* (see Appendix C for the EVK code). The routine *program_TRF6900* programs the settings into the TRF6900 via the serial interface. Figure 14 shows the settings for the C-word sent from *program_send_FSK*.

C-Word Settings of *program_send_FSK*



23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
Address Bits			APLL			NPLL	MM	SLCTL			PLL	VCO	PA		SLC	LPF	SW	RSSI	DEM	IF	MIX	LNAM	

Figure 14. C-Word Settings Programmed by *program_send_FSK*

The C-word register contents are shown in Table 1

Table 1. Contents of C-Word Register

CONTROL BIT NAME	BIT(S)	SETTING	DESCRIPTION	SELECTED
APLL	[18]–[20]	111	Acceleration factor for the charge pump	140
NPLL	[17]	0	RF divider ratio for PLL	256
MM	[16]	0	Modulation mode select	FSK
SLCTL	[15]	1	Slicer mode select bit	Learning
Not used	[13], [14]			
PLL	[12]	1	Enable PLL	Enable
VCO	[11]	1	Enable VCO	Enable
PA	[9], [10]	11	Power amplifier gain value	0 dB
SLC	[8]	0	Enable data slicer	Disabled
LPF	[7]	0	Enable LPF amplifier	Disabled
SW	[6]	0	Data switch	Demodulator
RSSI	[5]	0	Enable limiter/RSSI	Disabled
DEM	[4]	0	Enable limiter/demodulator	Disabled
IF	[3]	0	Enable first IF amplifier	Disabled
MIX	[2]	0	Enable mixer	Disabled
LNA	[0], [1]	00	Low noise amplifier operation mode	Disabled

The RF link implemented on the EVK uses Mode 1 for sending and requires all the components on the receiver side of the TRF6900 in C-word to be disabled.; on the other hand, the components for the transmission are all active. The following settings are noted: the power amplifier sends with maximum output power; the blocks of the PLL are all active; for modulation, FSK is selected; the RF divider ratio is 256 (chosen because with the 18-MHz reference crystal, spurious signals are forced outside the operating spectrum). The acceleration factor for the charge pump has been set to the maximum value of 140. The slicer control bit was set to learning mode because this is the default setting necessary to detect the training sequence of the sender. See Section 4.2 for a detailed explanation.

The next setting for C-word is shown in Figure 15. The only difference from the setting already described is the change of the data-slicer mode from learn to hold mode. This setting is used for the reception of the data package in hold mode to enable the reception of NRZ-encoded signals. The mechanism is as described in Section 3.2.1.

C-Word Settings of *program_receive_FSK_hold*

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
Address Bits			APLL			NPLL	MM	SLCTL			PLL	VCO	PA		SLC	LPF	SW	RSSI	DEM	IF	MIX	LNAM	

Figure 15. C-Word Settings Programmed by *program_receive_FSK_hold*

The C-word contents are valid for Mode 1, i.e., the second mode for the EVK. However, as shown in Figure 10, there are also control bits valid for both operating modes. Therefore the C-word must be reprogrammed to switch the data slicer from learn- to hold-modes during reception.

The only setting for the D-word used for the RF link on the EVK is programmed by *program_receive_FSK_learn*. The D-word is used for Mode 0; therefore, all the control bits uniquely valid for Mode 0 are programmed for reception. Thus, all components used by the receiver path of the TRF6900 are enabled in the D-word. Figure 16 shows the actual settings.

D-Word Settings of *program_receive_FSK_learn*

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	1	1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	1	1	1
Address Bits			DEV								PLL	VCO	PA		SLC	LPF	SW	RSSI	DEM	IF	MIX	LNAM	

Figure 16. D-Word Settings Programmed by *program_receive_FSK_learn*

The D-word contains (i) settings valid only for Mode 0, and (ii) a deviation setting with mode-independent validity when FSK is selected. The deviation setting is added to the actual DDS setting when TXDATA goes high. The deviation has been set to 60 kHz for the EVK.

To demonstrate their usage, all available modules have been used for the RF link implemented on the EVK, except the RSSI. Table 2 gives an overview of the settings.

Table 2. D-Word Settings

CONTROL BIT NAME	BIT(S)	SETTING	DESCRIPTION	SELECTED
DEV	[13]–[20]	111000	FSK frequency deviation register	60 kHz
PLL	[12]	1	Enable PLL	Enable
VCO	[11]	1	Enable VCO	Enable
PA	[9[, [10]	0	Power amplifier gain value	Disabled
SLC	[8]	1	Enable data slicer	Enabled
LPF	[7]	1	Enable LPF amplifier	Enabled
SW	[6]	0	Data switch	Demodulator
RSSI	[5]	0	Enable limiter/RSSI	Disabled
DEM	[4]	1	Enable limiter/demodulator	Enabled
IF	[3]	1	Enable first IF amplifier	Enabled
MIX	[2]	1	Enable mixer	Enabled
LNA	[0], [1]	11	Low noise amplifier operation mode	Normal operation

The low noise amplifier (LNA) runs with maximum gain in normal operation and the mixer is enabled. The received frequency of 869.83 MHz must be mixed down to the intermediate frequency (IF) of 10.7 MHz. The application example on the EVK board uses two identical ceramic filters; therefore, the first IF amplifier is switched on. The purpose of the IF amplifier is to decouple the IF filters and to compensate for filter losses. The demodulator recovers the low-frequency signal from the IF signal by demodulating the IF signal. The demodulator is a simple quadrature demodulator and delivers a signal adequate for the frequency of the IF signal. The low pass filter (LPF) is used to filter high frequency noise from the demodulator output signal. Finally, the data slicer recovers the digital baseband from the LPF output which is then processed by the microcontroller.

4.2 Detection of the Training Sequence

The RF link (TRF6900 Demo 1.16) implemented on the MSP-EVKTRF6900 uses a single frequency. When the EVK is in system mode, following power-on-reset, the EVK is in receive mode by default. Thus, the MSP430 on the EVK board programs the TRF6900 for reception in learn mode at the selected frequency. The TRF6900 is controlled by the MSP430 and is switched to active mode. The MSP430 scans the received signal for a valid data package.

Because the band always contains noise, the data slicer output (DATA_OUT) necessarily always delivers a digitized signal component. Another cause for a signal at the data slicer output is another application using the same frequency band. In any case, the TRF6900 always delivers a signal to RXDATA when it is active and in receive mode. The MSP430 must distinguish between valid and invalid signals.

The MSP430 uses the Timer_A module to make this distinction. The clock source for Timer_A is the external high-frequency crystal so Timer_A is clocked at 2.4576 MHz. The selected mode is the continuous-up mode. In this mode Timer_A counts continuously from 0 to FFFFh, then generates an interrupt and restarts. The RXDATA signal is applied to a capture-compare block. Figure 17 shows how every edge of the RXDATA signal generates an interrupt and triggers capture of the current value of Timer_A for storage. The next interrupt from RX-DATA restarts the same procedure. The newly-captured value of Timer_A is compared with the previous one and, in this way, the MSP430 calculates the width of the received pulse. Because the transmission of a data package always starts with the same training sequence (pulse width 26.04 µs), the MSP430 scans the signal on the RX-DATA line for pulses of 26.04 µs duration.

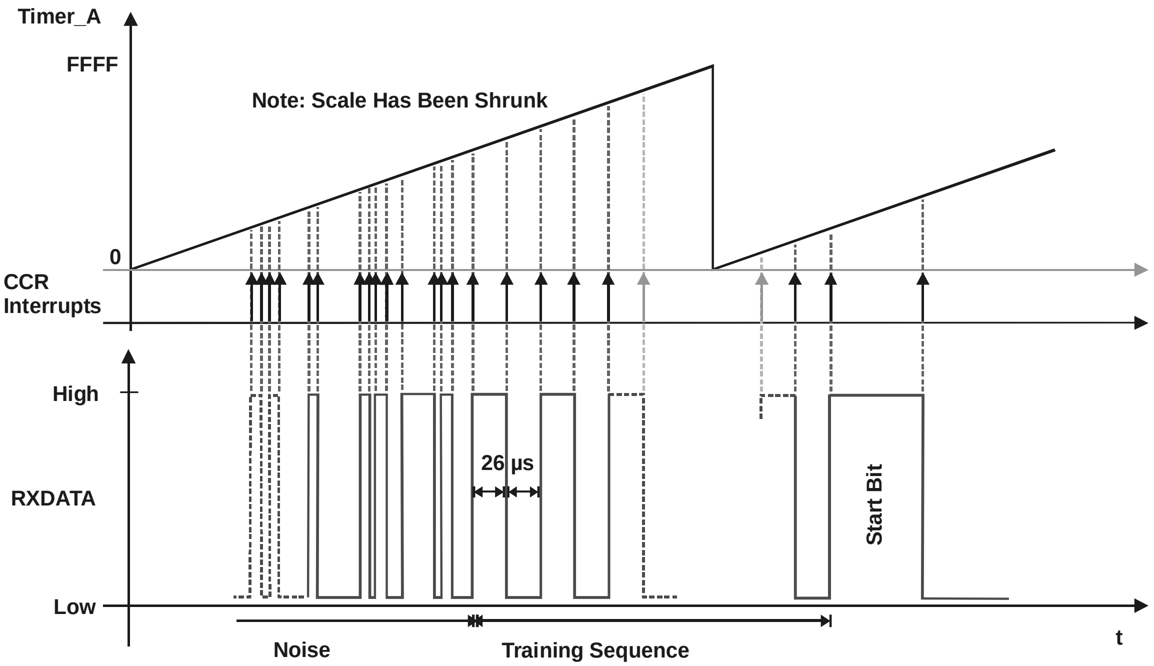


Figure 17. Scanning RXDATA—Timer_A/CCR Interrupts

Because the received signal is distorted by noise within the RF band as well as by other effects, the RXDATA signal always contains jitter, as shown in Figure 18. Figure 18 shows oscilloscope traces of the TXDATA signal on the transmitter side and the corresponding RXDATA signal on the receiver side. This figure shows that, because of jitter, the MSP430 scans the received signal for a duration between 22.8 µs and 29.3 µs, rather than for precisely 26.04 µs. This range is empirical and results from tests performed during the evaluation phase. The corresponding number of clock cycles is from 56 to 72.

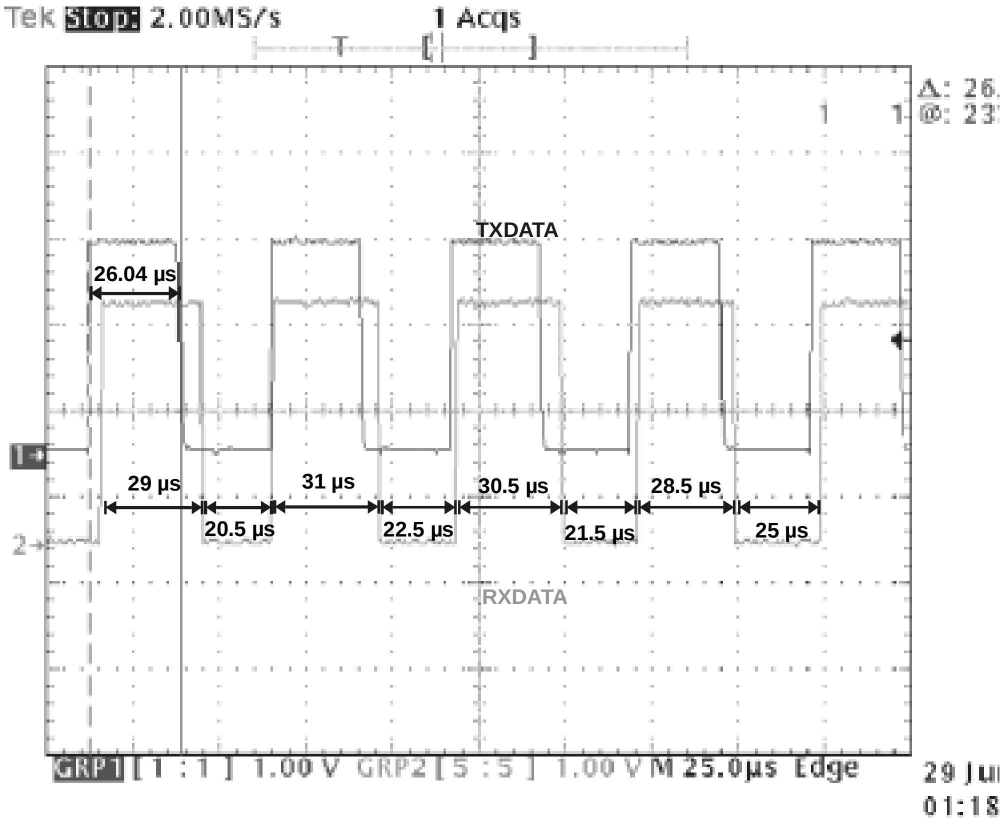


Figure 18. Jitter of the RXDATA Signal—RXDATA Compared With TXDATA

As implemented on the MSP-EVKTRF6900, recognition by the training sequence is somewhat adaptive. The MSP430 does not scan the received signal on a fixed time grid; instead, it scans every individual pulse on the RX-DATA line.

Determining an acceptable tolerance for distinguishing between valid and invalid data is application dependent. The compromise is between loosing or failing to recognize a valid bit on the one hand, and accepting an invalid bit as valid, on the other. Scanning of the RXDATA signal is handled by an interrupt routine (*CC2_INT*) that contains a jump table (see code in Appendix O). In recognizing the training sequence, the code distinguishes between three different states:

- No valid pulse is detected.
- Fewer than 8 valid consecutive pulses are detected.
- More than 8 valid consecutive pulses are detected—the training sequence is recognized.

The interrupt routine for the detection of the training sequence has 5 possible results is shown in Figure 19. Recognition of the training sequence is achieved when the MSP430 recognizes 8 consecutive valid pulses. The probability of a single valid pulse being received by accident is quite high because there is high probability of a received RXDATA signal being caused by noise or by other senders in the RF band. Therefore it is only a question of time before a pulse recognized as valid is detected, even when no valid data package has been transmitted. The requirement of 8 valid successive pulses (no intervening invalid pulse) reduces the probability of validating an invalid signal to a tolerable level for this application example.

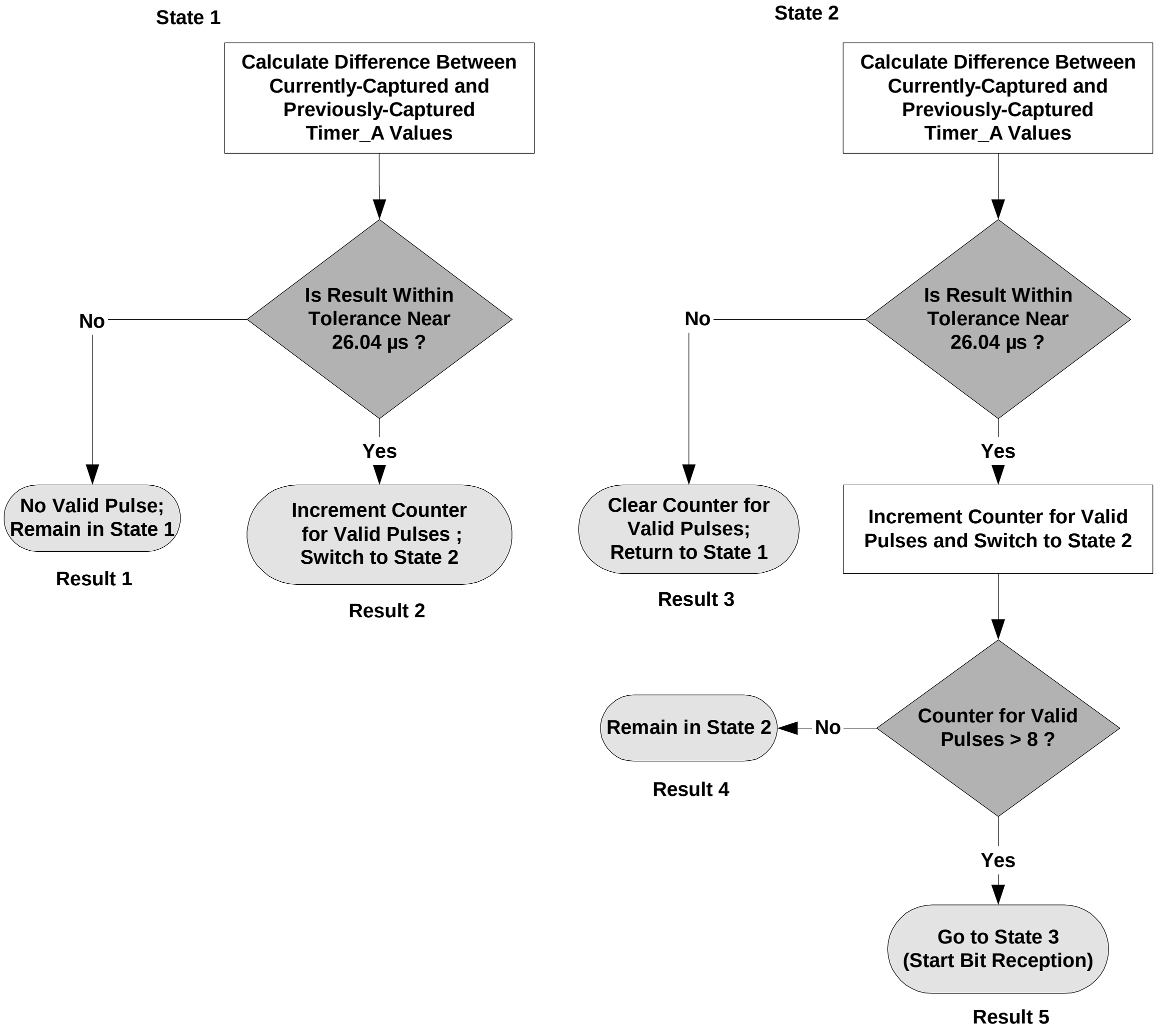

State 1
Calculate Difference Between Currently-Captured and Previously-Captured Timer_A Values
Is Result Within Tolerance Near 26.04 µs ?
No
Yes
No Valid Pulse; Remain in State 1
Result 1
Increment Counter for Valid Pulses ; Switch to State 2
Result 2
State 2
Calculate Difference Between Currently-Captured and Previously-Captured Timer_A Values
Is Result Within Tolerance Near 26.04 µs ?
No
Yes
Clear Counter for Valid Pulses; Return to State 1
Result 3
Increment Counter for Valid Pulses and Switch to State 2
Counter for Valid Pulses > 8 ?
No
Remain in State 2
Result 4
Yes
Go to State 3 (Start Bit Reception)
Result 5

Figure 19. Flowchart for Detection of the Training Sequence, *CCR2_INT*

4.3 Detection of the Start Bit

The principle of start-bit recognition is similar to the detection of the training-sequence pulses—the difference is, that the start bit consists of a single pulse. To distinguish between the pulses of the training sequence and the pulse of the start bit, the length of the start bit was set to three times 26.04 µs, i.e., three times as long as the pulses of the training sequence. The falling edge of the start-bit pulse is the trigger for the scanning and reading of the data package.

4.4 Reading the Data Package

The data package immediately follows the start bit. It contains 32 bytes of data plus 2 bytes for the checksum of the transmitted data package. The first two bytes after the start bit contain the data-package checksum. The pulse code modulation (PCM) used for the data package is NRZ. The single-bit pulse width is 26.04 µs and corresponds to a 38.4 kbps data transmission rate.

The procedure for scanning the data package differs from detection of the training sequence and the start bit. This is because the information in the data package is stored, not in the pulse width of the signal (as it is during the training sequence), but in the logic level on the base-band side, or a frequency value on the RF side. Because the pulse width (one bit) and the number of transmitted bits are known, it is more efficient to check the value of the data-slicer output signal on a defined time interval (26.04 µs) than to check the pulse width and the value of the signal. The scan trigger is the falling edge of the start bit. Figure 20 shows that after a certain delay (used for setting up the microcontroller settings), the microcontroller starts scanning the data package. The microcontroller checks the logic value of the data-slicer mid-bit output signal. Because the only synchronization for the scan is the falling edge of the start bit on the receiver side, the jitter of that edge and the accuracy of the microcontroller clocks on both sides can profoundly affect errors generated during data reception. Therefore, to achieve the maximum transmission range, it is necessary in some cases to synchronize several times, or to implement a more complex higher-accuracy synchronization. Although the jitter of RXDATA rises when the field strength of the received signal approaches the limit of receiver sensitivity, tests have shown that this simple synchronization is sufficient for acceptable performance of the implemented link.

It may also be advantageous to scan several times during a bit pulse. However, measurements on the EVK have shown this to have an impact only at the boarder of the range, where the signal to noise ratio becomes limiting in any case.

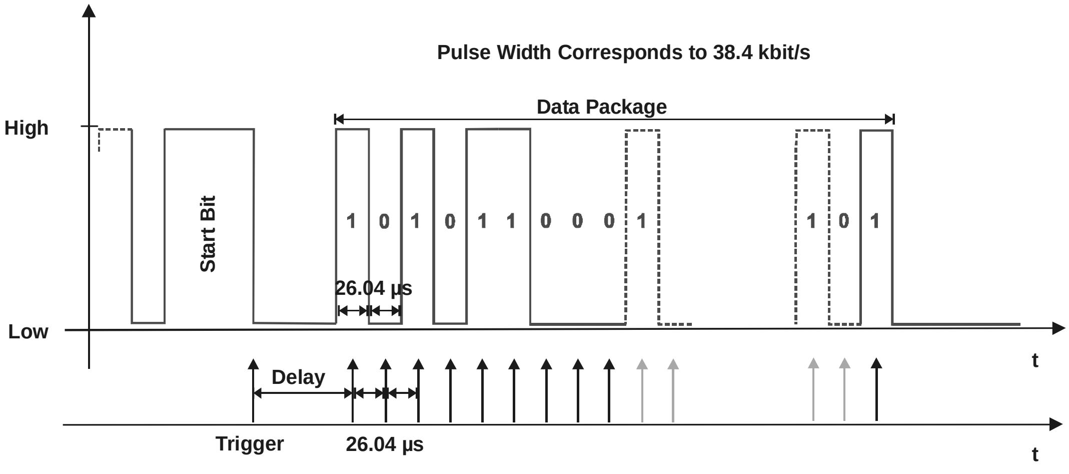


Figure 20. Scanning the Data Package

4.5 Generating the Time Grid for Scanning the Data Package

The description of the procedure used for scanning the data package indicated that the timing of the scan process is decisive to the error rate on the receiver side, so exact timing generation is essential. Certainly an accurate microcontroller clock is essential; however, it is also possible to generating exact timing in a very simple way. For this purpose, the Watchdog Timer (WDT) of the microcontroller is used in timer mode. The WDT in timer mode generates Interrupts at fixed intervals (depending on the WDT clock source and the selected prescalers) without causing a power-on-reset (POR).

Data package scanning is done in a loop that tests the signal of RXDATA in the middle of the expected pulse, stores the value, increments counters, and checks whether there are more bits to receive; if there are, it forces the microcontroller into a sleep mode. The periodic awakening is timed by the WDT interrupt which also sets the signal-scan timing. The interrupt routine simply resets the sleep mode of the MSP430.

5 Implementation of RF Transmission

Transmission of an RF signal is much simpler on the baseband side than its reception. The effort needed for the RF design of the transmitter (PLL, VCO setup) far exceeds that needed to generate the baseband signal for RF transmission. This is covered in another application report (SWRA033).

The microcontroller's task in the transmission process is to receive the data package from the PC, then generate the PCM signal for the RF data package. The MSP430 switches the TRF6900 to Mode 1 (transmit mode). The microcontroller is also responsible for generating the timing for the transmission protocol, which is identical to the timing for scanning the data package. The cycle is as follows: In timer mode, the WDT awakens the microcontroller from sleep mode (the CPU being switched off), the microcontroller executes the code (e.g., it sets TXDATA according to the bit to be transmitted), and a sleep instruction follows.

5.1 Generating the Training Sequence

The purpose and form of the training sequence have been described in section 4.2. Its length is 38 pulses, each of 26.04 µs duration, and this results in a training sequence of about 990 µs. As already mentioned, pulse timing for the training sequence is generated by the WDT in timer mode. In a cyclic loop, the TXDATA signal is toggled, the counter for the pulses of the training sequence is checked and decremented, and the MSP430 is switched into sleep mode (by turning off the CPU). Awakening is done every 26.04 µs by an interrupt from the WDT in timer mode (see Figure 21).

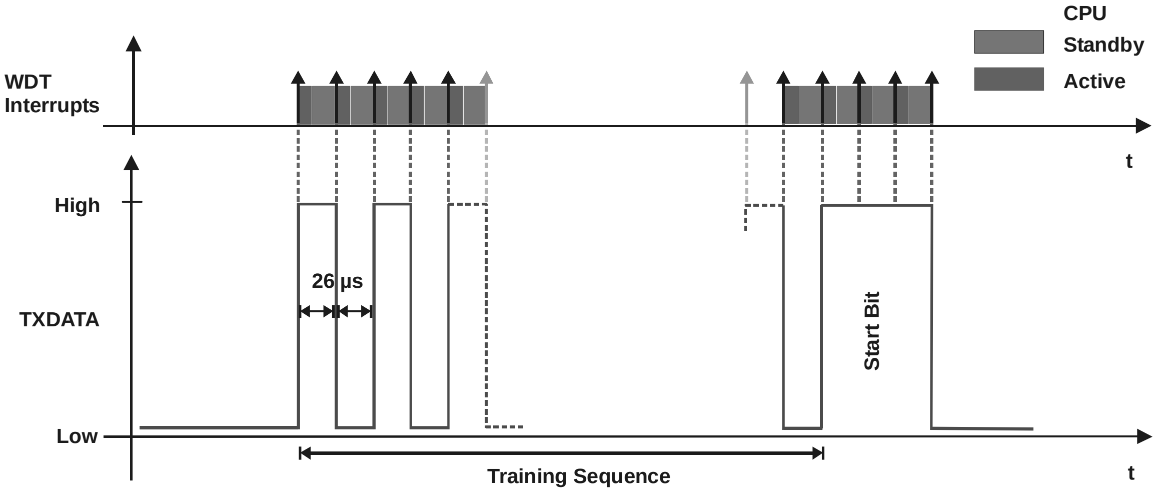


Figure 21. Generation of the Training Sequence—Baseband

5.2 Generation of the Start Bit

The procedure to generate the start bit is the same as for the training sequence (see Figure 21). Instead of toggling the TXDATA signal at every WDT interrupt, the MSP430 toggles the line after the third interrupt.

5.3 Sending the Data Package

Timing for data package transmission is the same as for the training sequence and is initiated by the start bit from the WDT in timer mode. Because the data package is sent as NRZ code, further processing or encoding of the base-band signal is not needed. The first two bytes of the data package contain the checksum of the data to be transmitted. This checksum is calculated after reception from RS232 and storage of the data package in RAM (for the location see Appendix I). The checksum enables the receiver to confirm the validity of the data it received. In the transmission procedure, bytes are consecutively shifted bit-wise left through carry, with the MSB always transmitted first. Depending on the state of carry, the TXDATA signal is set. There is no start or stop bit between the single bytes or words. The transmitted bytes are sent sequentially and continuously. Figure 22 shows the timing diagram.

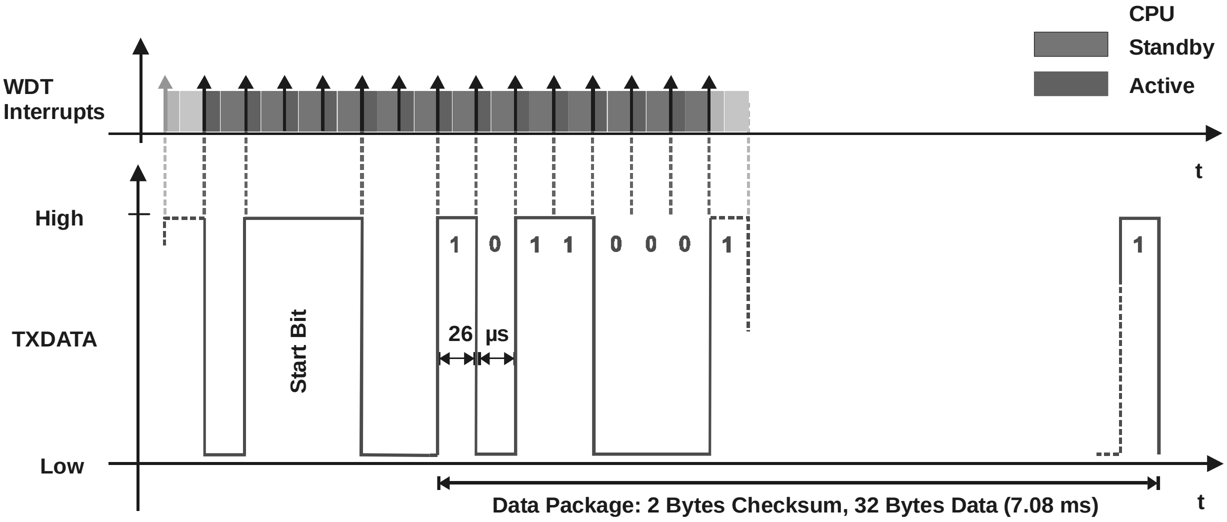


Figure 22. Generation of the Data Package—Baseband

6 Hardware for the RF Link

The RF link implemented is based on the MSP-EVKTRF6900. The parts list, which can be found in *TRF6900/MSP430 EVK* (SWRA032), is extensive but can be downsized for a dedicated link. For example, all the connectors used for making measurements, the various links to the PC, and the voltage regulator are unnecessary for a functional RF link. Figure 23 shows the connections between the TRF6900 and the MSP430.

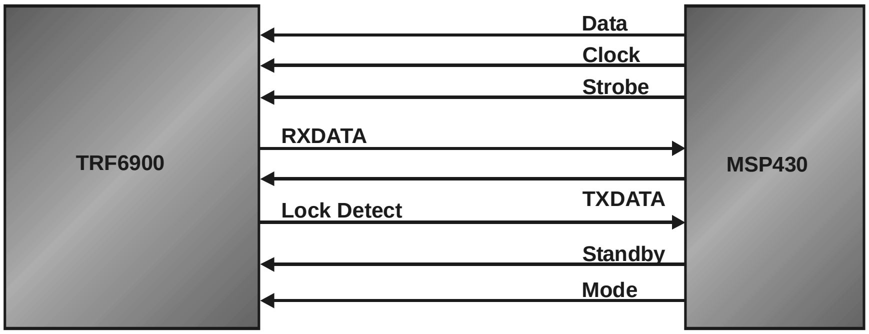


Figure 23. Interface TRF6900—MSP430

For programming the four frequency and control registers, the TRF6900 has data, clock and strobe connectors. In addition there are two connectors for the transmitted (TXDATA) and received (RXDATA) data signals. RXDATA delivers the received, demodulated and filtered data-slicer output signal. TXDATA is the line to which the MSP430 applies the base-band output signal. Toggling the logic level on the TXDATA line (using FSK) forces the TRF6900 to switch between the DDS and (DDS + deviation) frequencies. The TRF6900 uses the lock-detect line to signal that the PLL is locked to the selected frequency. When the lock detect signal is set and the settings for the other needed modules (RX, TX, PA, LNA, etc.) are correct, then the device is ready for transmission or reception, considering the run-in times of other activated modules.

The standby and mode control lines are used for quickly switching the TRF6900 to standby (low power consumption, 2 µA) on the one hand, and for fast switching between the two preprogrammed modes (Mode 0 and Mode 1).

In addition to the digital signals, the TRF6900 also delivers two analog signals. These are not used by the MSP430; rather, the first is the received signal strength indicator (RSSI) output, and the second is the analog output of the post-detection amplifier, which is the input signal to the data slicer.

Schematics for the MSP-EVKTRF6900 can also be found in *TRF6900/MSP430 EVK* (SWRA032).

7 Code Description

Appendices A to R contain the entire code used for implementing the RF link. The code is in assembly language for the MSP430F1121 and runs in the IAR kickstart environment. A version for the TI simulation environment is also available. The code description consists of flow charts and code listings.

The code description also contains IIC routines for an IIC-LCD. These routines are not documented with flowcharts. They are used to display the received data on the LCD. The Gerber files and schematics for the additional board for the LCD are available on the CD-ROM for the MSP430.

Reference

TRF6900/MSP430 EVK , Application Report, Texas Instruments Literature Number SWRA032

Appendix A. *main*

The main routine consists of two parts: initialization, which is executed only once after POR, and the main loop, set up as a closed loop (see Figure A-1).

In the initialization phase, the settings for the ports, the modules, and the clock systems are calculated (see code for detailed settings). The *loop_main* routine provides the RS232 and RF communication and includes calculation of checksums, and the IIC routines for the LCD used to display the received data.

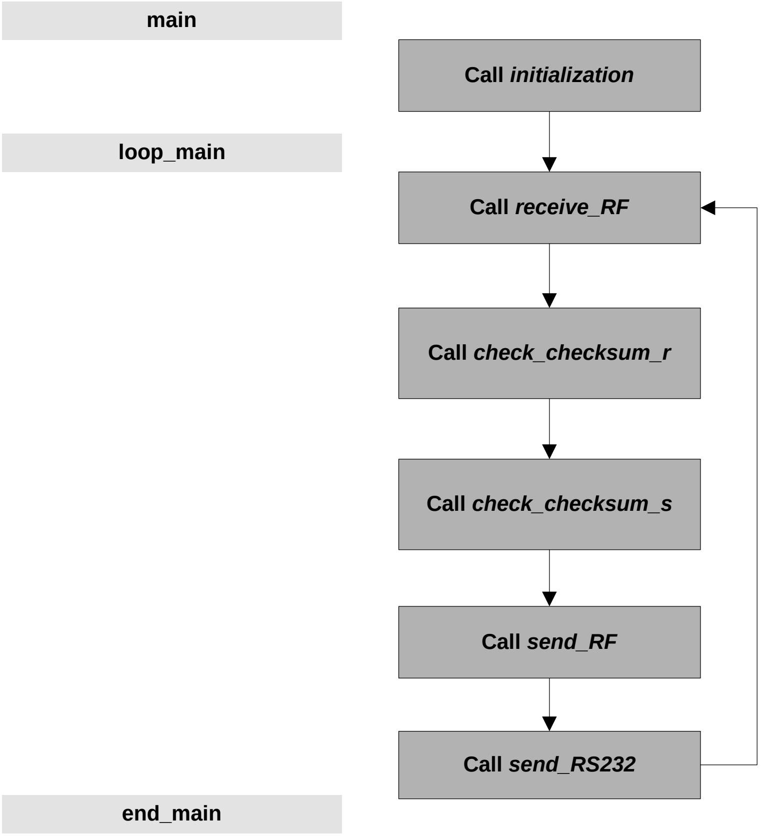


Figure A-1. **Logic Diagram for *main***

```
;*******************************************************************************************************************;
MSP430F1121 Program for the MSP-EVKTRF6900 Revision B1
;
; This code supports the transmission and reception of 32 data bytes
; and 2 checksum bytes.
; It supports an acknowledge.
; Displays the received data on an IIC-LCD
; The data transmission rate is 38.4 kbit/s on the RF side
; The data transmission rate at the RS232 port is 19.2 kbit/s
;
; Written by:   Peter Spevak
; Texas Instruments Deutschland GmbH
; June 2000
; Last change:  19.6.2000
;*****************************************************************************************************************

#include    "TRF6900_1121.H"            ; include the special settings for
                                        ; MSP-EVKTRF6900

RAM_orig     EQU       00200h           ; RAM start
FLASH_orig   EQU       0F000h           ; F1121 FLASH start

NAME       TRF6900_Demo

;********************************************** Main ********************************************************
; last change:    April 12th 2000             Last print: November 19th 1999
;
; calls all the subroutines
; is the frame for the whole program
;
;
;*****************************************************************************************************************

            RSEG      CODE                      ; beginning of the code
START

main
            MOV       #0300h,SP                 ; initialize system stack pointer
            DINT                                ; general Interrupt disable
            MOV       #WDTPW+WDTHOLD,&WDTCTL
                                                ; write with password 5A00h, WDT off
            CALL      #initialization           ; initialization of the TRF6900 and
                                                ; MSP430 settings
            CALL      #IIC_init_EVK_string      ; load the initial string to RAM
            CALL      #IIC_internal_voltage     ; initialize the internal voltage generator
            CALL      #IIC_initialize_LCD       ; initialize the LCD
            CALL      #IIC_LCD_display          ; IIC-routine for LCD display


loop_main
            CALL      #receive_RF               ; receive 10x8bits
            BIC       #TAIE,&TACTL              ; disable Timer_A interrupt
            BIC       #CCIE,&CCTL1              ; disable CCR1 interrupt
            CALL      #check_checksum_r         ; check the checksum on the receiver side
            CALL      #check_checksum_s         ; check the received checksum from receiver
            MOV       #099Ch,wait_r             ; init. wait_r for 250µs, to delay the
                                                ; acknowledge, run in time of the sender
                                                ; from receive to send mode
            CALL      #wait_x_cycles            ; wait 250µs
            CLR       counter                   ; set counter value to 0
            CALL      #send_RF                 ; send acknowledge
            CALL      #rs232_send               ; transmits the received data via RS232 Port
            CALL      #IIC_initialize_LCD       ; initialize the LCD
```

```
        CALL    #IIC_LCD_display     ; IIC-routine for LCD display
        BIC     #CCIFG,&CCTL1        ; reset interrupt flag
        BIC     #TAIFG,&TACTL        ; reset interrupt flag
        BIS     #TAIE,&TACTL         ; enable interrupt
        BIS     #CCIE,&CCTL1         ; enable interrupt
        JMP     loop_main            ; end of the main routine

end_main
```

Appendix B. *Initialization*

The purpose of the *initialization* routine is to configure the ports, modules, and the basic clock system of the MSP430F1121 for the RF link. The routines that are used to program the four words of the TRF6900 (to initialize the TRF6900) are called from *initialization*. This prepares the TRF6900 for data reception and enables fast switching from receive to transmit modes.

Initializing the basic-clock module of the MSP430F1121 needs special attention. Just like other members of the MSP430x1xx family, the basic-clock module features automatic switching of the clock source for the CPU back to (digitally controlled oscillator (DCO) mode when the oscillator fault interrupt flag (OFIFG) is set. This happens, when the external clock source fails, or on every power-on-reset. Therefore, switching from the DCO as clock source for the CPU to an external high-frequency crystal (in the case of the EVK, for example), must be done by a defined procedure. This is handled by the oscillator-fault-interrupt-flag loop in the initialization routine. The steps are as follows:

1. This step occurs outside the loop. It consists of switching on the high frequency oscillator, which needs a finite time for start-up.
2. This step takes place inside the loop and resets the OFIFG to POR.
3. Because, apart from the POR, the OFIFG is set after approximately 50 missing cycles, the third step is a wait routine, used to give the oscillator fault detection logic enough time to ensure that the OFIFG will not be set after leaving the loop.
4. Test the OFIFG. If the OFIFG is not set, the oscillator has stabilized, and can be used for the CPU. The selection of the high frequency clock for the CPU is done outside after the loop.

When OFIFG is set, using code to force the microcontroller to switch to the external crystal can potentially result in code execution errors.

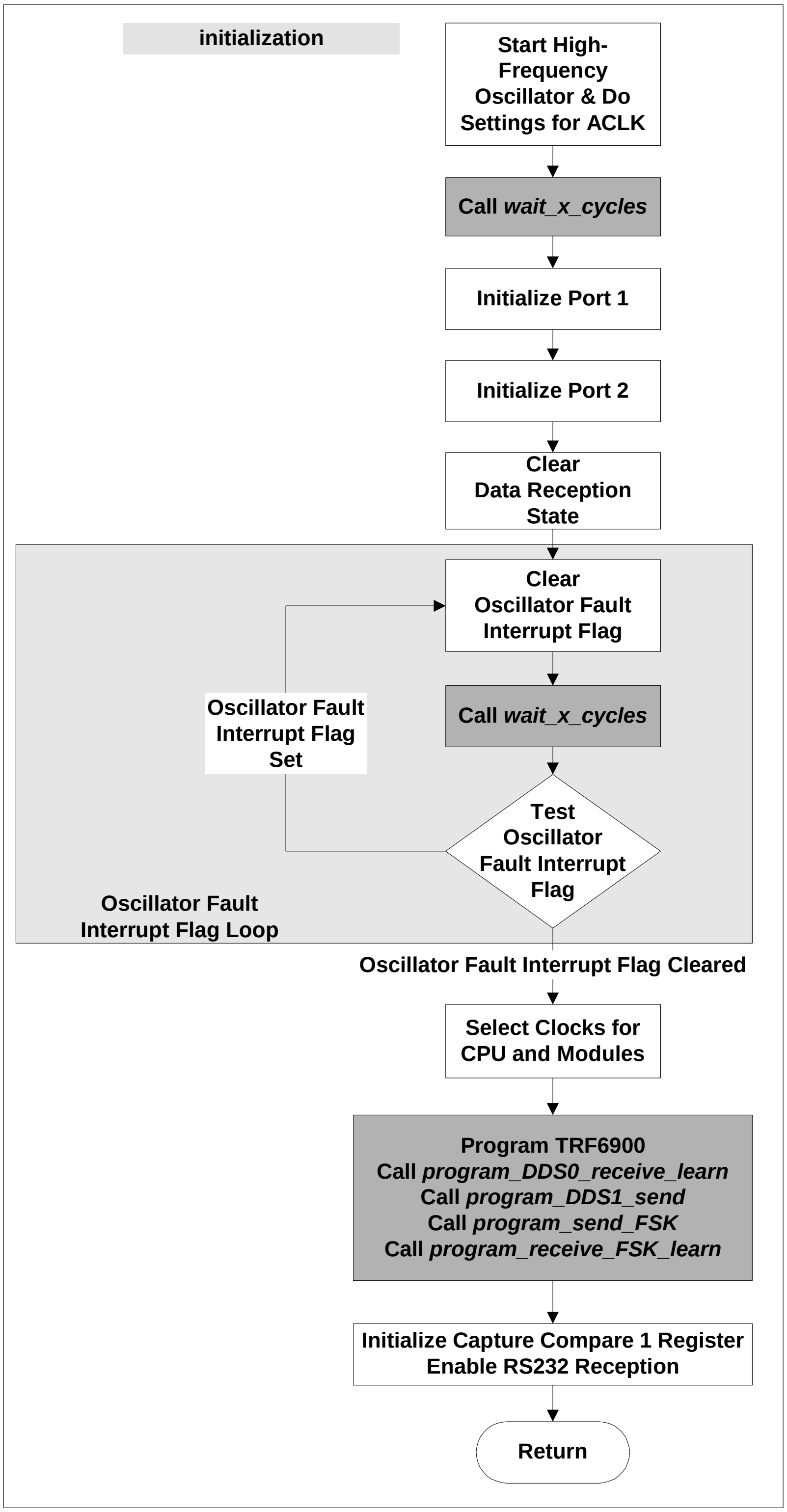

initialization
Start High-Frequency Oscillator & Do Settings for ACLK
Call wait_x_cycles
Initialize Port 1
Initialize Port 2
Clear Data Reception State
Clear Oscillator Fault Interrupt Flag
Call wait_x_cycles
Test Oscillator Fault Interrupt Flag
Oscillator Fault Interrupt Flag Set
Oscillator Fault Interrupt Flag Loop
Oscillator Fault Interrupt Flag Cleared
Select Clocks for CPU and Modules
Program TRF6900
Call program_DDS0_receive_learn
Call program_DDS1_send
Call program_send_FSK
Call program_receive_FSK_learn
Initialize Capture Compare 1 Register
Enable RS232 Reception
Return

Figure B-1. Logic Diagram for *initialization*

```
;************************************** Initialization ****************************************************
;   last change: June 19th 2000                  Last print: November 19th 1999
;
;   affects:     clock settings
;                initializes the hardware
;                clearing oscillator fault flag !!! necessary for MSP430F1121
;**********************************************************************************************************

initialization

;********************************** Clock Settings ********************************************************
; 1. LFXT1CLK-signal is driven by the external 2.4576Mhz crystal
; 2. ACLK is 1/8*LFXT1CLK
; 3. MCLK directly coupled to LFXT1CLK (used only for the CPU)
; 4. SMCLK directly coupled to LFXT1CLK (used for Timer_A during code reception)
;**********************************************************************************************************
            MOV.B     #clock_new_1,&BCSCTL1  ; set the divider for ACLK to 8
                                             ; high frequency oscillator selected
                                             ; oscillator on
            MOV       #0FFFh,wait_r          ; for programming purpose
                                             ; MSP430P112 and oscillator start up
            CALL      #wait_x_cycles         ; wait routine, waits 3+x*13 cycles

initialize_port1
            MOV.B     #txd,&P1OUT            ; set TXD_MSP(P1.1), TXD_MSP by default
                                             ; high
                                             ; reset TXDATA(P1.4), DATA(P1.7),
                                             ; CLK(P1.6), STROBE(P1.5)
            MOV.B     #stdb_rs232+data+clk+tx+txd,&P1DIR
                                             ; switch DATA(P1.7), CLK(P1.6),
                                             ; TXDATA(P1.4), TXD(P1.1),
                                             ; stdb_rs232(P1.0) to output dir.
            MOV.B     #rx+rxd,&P1SEL         ; select the module function for P1.3 (TA2),
                                             ; RXD P1.2 (TA1)

initialize_port2
            MOV.B     #LED3,&P2OUT           ; reset P2.0,CTS (-> CTS  set!),
                                             ; STANDBY_TRF6900(P2.5), MODE(P2.1) ->
                                             ; receive, set LED3(P2.3), System Mode
            MOV.B     #stdb_trf6900+mode+cts+LED3,&P2DIR
                                             ; switch STANDBY_TRF6900(P2.5),
                                             ; MODE(P2.1), CTS(P2.0) to output,
                                             ; LOCKDET(P2.4) input
initialize_receive_status
            CLR       &data_rx_state         ; reset receive status

oscillator_flag
            BIC.B     #02h,&IFG1             ; reset the oscillator fault flag
            MOV       #0Fh,wait_r            ; wait 3 + Fx13 cycles
            CALL      #wait_x_cycles         ; call the wait routine
            BIT.B     #02h,&IFG1             ; test the oscillator fault flag
            JNZ       oscillator_flag
            BIC.B     #02h,&IFG1             ; clear oscillator fault flag
```

```
initialize_Clock

            MOV.B       #clock_new_2,&BCSCTL2  ; SELS = 1 LFXT1CLK source for SMCLK
                                               ; LFXT1CLK source for MCLK

;------------------------------------------------------------------------------------------------------------------------------
                                               ; programming the 4 words to TRF6900
            CALL        #program_DDS0_receive_learn
            CALL        #program_DDS1_send
            CALL        #program_send_FSK
            CALL        #program_receive_FSK_learn

initialize_CCR1_interrupt

            MOV         #CCIE+CAP+CMNEG,&CCTL1
                                               ; interrupt enable, capture mode, neg. edge
                                               ; for RS232 reception
end_intialization
         RET
```

Appendix C. *program_TRF6900*

Five different programming routines are used for the implemented RF link on the EVK. They are used to program the four words of the TRF6900. All of these routines have the same structure, they initialize two buffer registers with the bits to be programmed, and then call the actual programming routine called *program_TRF6900*, which executes the actual programming of the TRF6900 registers. Because of their similarity, only one of these routines is documented with a flowchart. The others are documented only within the code.

The *program_TRF6900* routine contents one special feature. It's the possible suppression of the strobe pulse, that activates the settings pushed into the stack of the TRF6900. This enables a fast switching of the TRF6900 from reception in learn to reception in hold mode.

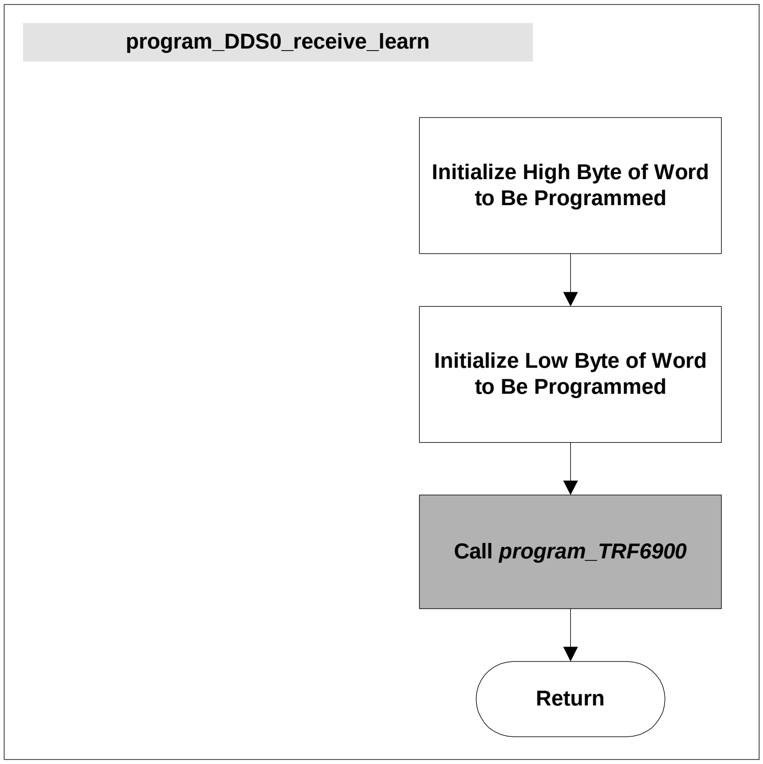

program_DDS0_receive_learn
Initialize High Byte of Word to Be Programmed
Initialize Low Byte of Word to Be Programmed
Call *program_TRF6900*
Return

Figure C-1. Logic Diagram for *program_DDSO_receive_learn*

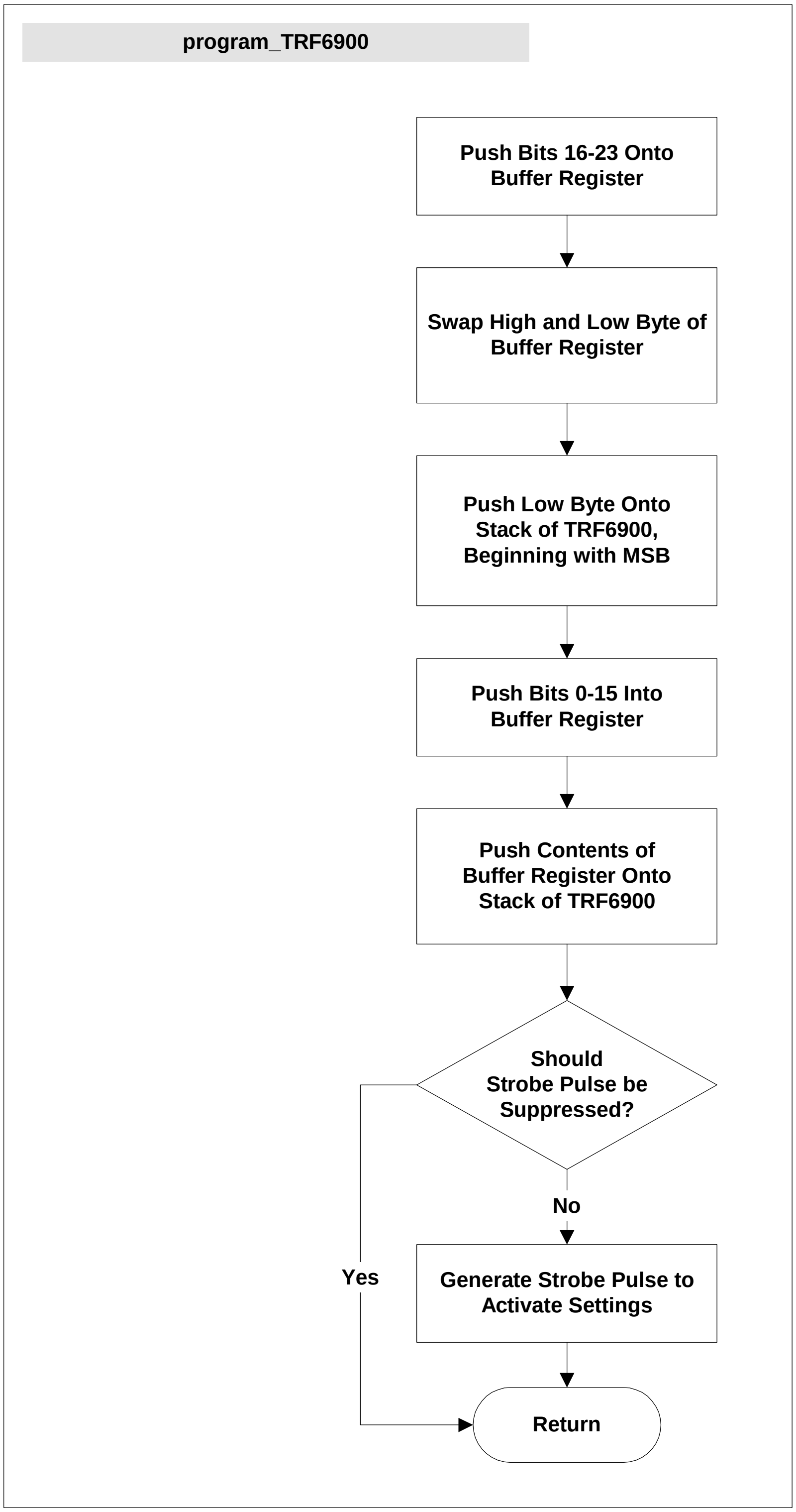

program_TRF6900
Push Bits 16-23 Onto Buffer Register
Swap High and Low Byte of Buffer Register
Push Low Byte Onto Stack of TRF6900, Beginning with MSB
Push Bits 0-15 Into Buffer Register
Push Contents of Buffer Register Onto Stack of TRF6900
Should Strobe Pulse be Suppressed?
No
Yes
Generate Strobe Pulse to Activate Settings
Return

Figure C-2. Logic Diagram for *program_TRF6900*

```
;********************************* Programming the TRF6900 ********************************************

;******************************* program_DDS0_receive_learn ******************************************
; last change: September 20th 1999
;
; purpose:    programs the DDS_0 for FSK Reception in learn mode (see further settings below)
;             (Mode 0, A-word)
;
;*******************************************************************************************************

program_DDS0_receive_learn
          MOV    #02Fh,word_h
          MOV    #0B920h,word_l                ; frequency for Mode 0, receive FSK in learn
                                               ; mode, DDS_0 settings for 859.13MHz,
                                               ; 18MHz system crystal, 10.7MHz IF
          CALL   #program_TRF6900

end_program_DDS0_receive_learn
          RET

;******************************* program_DDS1_send ****************************************************
; last change: September 20th 1999
;
; purpose:    programs the DDS_1 for Transmission (see further settings below)
;             (Mode 1, B-word)
;
;*******************************************************************************************************

program_DDS1_send
          MOV    #070h,word_h
          MOV    #05145h,word_l                ; frequency for Mode 1, send
                                               ; DDS_1 settings for 869.83MHz,
                                               ; 18MHz system crystal, 10.7MHz IF
          CALL   #program_TRF6900

end_program_DDS1_send
          RET

;******************************* program_send_FSK ****************************************************
; last change: September 20th 1999
;
; purpose:    programs the module Mode 1 for FSK transmission (see further settings below)
;             (Enable register for PLL, Data Slicer and Mode 1 settings, C-word)
;
;*******************************************************************************************************

program_send_FSK
          MOV    #0BCh,word_h
          MOV    #09E00h,word_l                ; Module Mode 1, send

; 1_LNAM  [bit 0..1]   00b   disabled          Low noise amplifier operation mode
; 1_MIX   [bit 2]      0b    disabled          Enable Mixer
; 1_IF    [bit 3]      0b    disabled          Enable 1st IF Amplifier
; 1_DEM    [bit 4]      0b     disabled          Enable Limiter/Demodulator
; 1_RSSI   [bit 5]      0b     disabled          Enable Limiter/RSSI
; 1_DSW    [bit 6]      0b     conn. to Demod.   Data Switch (Demodulator or RSSI)
; 1_LPF    [bit 7]      0b     disabled          Enable LPF Amplifier
; 1_SLC    [bit 8]      0b     disabled          Enable Data Slicer
; 1_PA     [bit 9..10]  11b    0dB att.          Power Amplifier gain value in TX mode
;                                                (see also MS bit)
; 1_VCO    [bit 11]     1b     enabled           Enable VCO
; 1_PLL    [bit 12]     1b     enabled           Enable PLL ( DDS System, RF divider,
;                                                Phase Comparator and Chargepump)
;          [bit 13..14] 00b                      not used
; SLCTL    [bit 15]     1b     learn mode        Slicer mode select bit
; MS       [bit 16]     0b     FSK               modulation mode select
; NPLL     [bit 17]     0b     256               RF divider ratio for PLL
; APLL     [bit 18..20] 111b   140               acceleration factor for the charge pump

        CALL    #program_TRF6900

end_program_send_FSK
        RET

;******************************** program_receive_FSK_learn **********************************************
; last change: September 20th 1999
;
; purpose:    programs the module Mode 0 for FSK reception in learn mode (see further settings
;             below)
;             (modulation and mode 0 settings, D-word)
;
;*****************************************************************************************************************

program_receive_FSK_learn
          MOV    #0C7h,word_h
          MOV    #0199Fh,word_l                  ; Modulation, Module Mode 0, receive

; 0_LNAM   [bit 0..1]   11b    normal          Low noise amplifier operation mode
;                              operation
; 0_MIX    [bit 2]      1b     enabled         Enable Mixer
; 0_IF     [bit 3]      1b     enabled         Enable 1st IF Amplifier
; 0_DEM    [bit 4]      1b     enabled         Enable Limiter/Demodulator
; 0_RSSI   [bit 5]      0b     disabled        Enable Limiter/RSSI
; 0_DSW    [bit 6]      0b     con. to Dem.    LPF Amplifier input routed to Demodulator
;                                              (FSK) or to RSSI (OOK)
; 0_LPF    [bit 7]      1b     enabled         Enable LPF Amplifier
; 0_SLC    [bit 8]      1b     enabled         Enable Data Slicer
; 0_PA     [bit 9..10]  00b    disabled        Power Amplifier gain value in TX mode (see
;                                              also MS bit)
; 0_VCO    [bit 11]     1b     enabled         Enable VCO
; 0_PLL    [bit 12]     1b     enabled         Enable PLL (DDS System, RF divider, Phase
;                                              comparator and Chargepump)
; MR       [bit 13..20] 0011 1000b             FSK frequency deviation register (corresponds with
;                                              60kHz modulation)

          CALL    #program_TRF6900
```

```
end_program_receive_FSK_learn
          RET


;****************************** program_receive_FSK_hold *********************************************
; last change: October 7th 1999
;
; purpose:    programs the module Mode 1 for FSK reception in hold mode (see further settings
;             below)
;             (enable register for PLL, Data Slicer and Mode 1 settings, C-word)
;
;*********************************************************************************************************

program_receive_FSK_hold
          MOV     #0BCh,word_h
          MOV     #01E00h,word_l                        ; Modulation, Module Mode 1, receive

; 1_LNAM  [bit 0..1]   00b    disabled           Low noise amplifier operation mode
; 1_MIX   [bit 2]      0b     disabled           Enable Mixer
; 1_IF    [bit 3]      0b     disabled           Enable 1st IF Amplifier
; 1_DEM   [bit 4]      0b     disabled           Enable Limiter/Demodulator
; 1_RSSI  [bit 5]      0b     disabled           Enable Limiter/RSSI
; 1_DSW   [bit 6]      0b     conn. to Demod.    Data Switch (Demodulator or RSSI)
; 1_LPF   [bit 7]      0b     disabled           Enable LPF Amplifier
; 1_SLC   [bit 8]      0b     disabled           Enable Data Slicer
; 1_PA    [bit 9..10]  11b    0dB att.           Power Amplifier gain value in TX mode
;                                                (see also MS bit)
; 1_VCO   [bit 11]     1b     enabled            Enable VCO
; 1_PLL   [bit 12]     1b     enabled            Enable PLL ( DDS System, RF divider,
;                                                Phase Comparator and Chargepump)
;         [bit 13..14] 00b                       not used
; SLCTL   [bit 15]     0b     hold mode         Slicer mode select bit
; MS      [bit 16]     0b     FSK                modulation mode select
; NPLL    [bit 17]     0b     256                RF divider ratio for PLL
; APLL    [bit 18..20] 111b   140                acceleration factor for the charge pump

        CALL    #program_TRF6900

end_program_receive_FSK_hold
        RET


;****************************** program_TRF6900 ********************************************************
; last change:    January 26th 2000
;
; purpose:    programs a word to A, B, C or D word register of the TRF6900
;             gets the settings from the calling routine in R6 and R7
;             suppresses the strobe pulse in receive_RF routine, to support shorter training
;             sequence
;
;*********************************************************************************************************

program_TRF6900
```

```
init_high_byte
        DINT
        BIC.B   #strobe,&P1OUT                  ; reset Strobe port
        BIS.B   #strobe,&P1DIR                  ; switch Strobe to output direction
        MOV     #02h,counter                    ; initialize the counter for high and low byte
        MOV     #08h,bits_r                     ; initialize bitcounter
        MOV     word_h,word_trf                 ; push the high byte to the programming buffer
        SWPB    word_trf                        ; push the low byte to the high byte, only the
                                                ; data in the low byte is relevant
        JMP     program_word

init_low_byte
        MOV     #010h,bits_r                    ; initialize bitcounter
        MOV     word_l,word_trf                 ; push the low byte to the programming buffer

program_word
        RLC     word_trf                        ; push the msb of the programming buffer to
                                                ; carry
        JNC     program_low

program_high
        BIS.B   #data,&P1OUT                    ; set data(P1.7)

program_clock
        BIS.B   #clk,&P1OUT                     ; generate a pulse on the clock line
        BIC.B   #clk,&P1OUT

program_next_bit
        DEC     bits_r                          ; decrement bit counter
        JNZ     program_word                    ; have already all bits been sent?
        DEC     counter                         ; decrement counter for low byte recognition
        JNZ     init_low_byte                   ; low byte is to be programmed

generate_strobe
        BIC.B   #data,&P1OUT                    ; reset data (P1.7)
        BIT     #08h,&data_rx_state             ; shall the strobe pulse be suppressed?
        JNZ     end_program_TRF6900             ; yes suppress it
        BIS.B   #strobe,&P1OUT                  ; set strobe(P1.5)
        BIC.B   #strobe,&P1OUT                  ; clear strobe(P1.5)
        BIC.B   #strobe,&P1DIR                  ; set strobe(P1.5) to input direction

end_program_TRF6900
        EINT
        RET                                     ; back to calling routine

program_low
        BIC.B   #data,&P1OUT                    ; clear data(P1.7)
        JMP     program_clock
```

Appendix D. *send_RF*

This routine is used to transmit the received data from the PC via the RF to its counterpart. It is also used to send the acknowledge after receipt of a valid (correct checksum) data package.

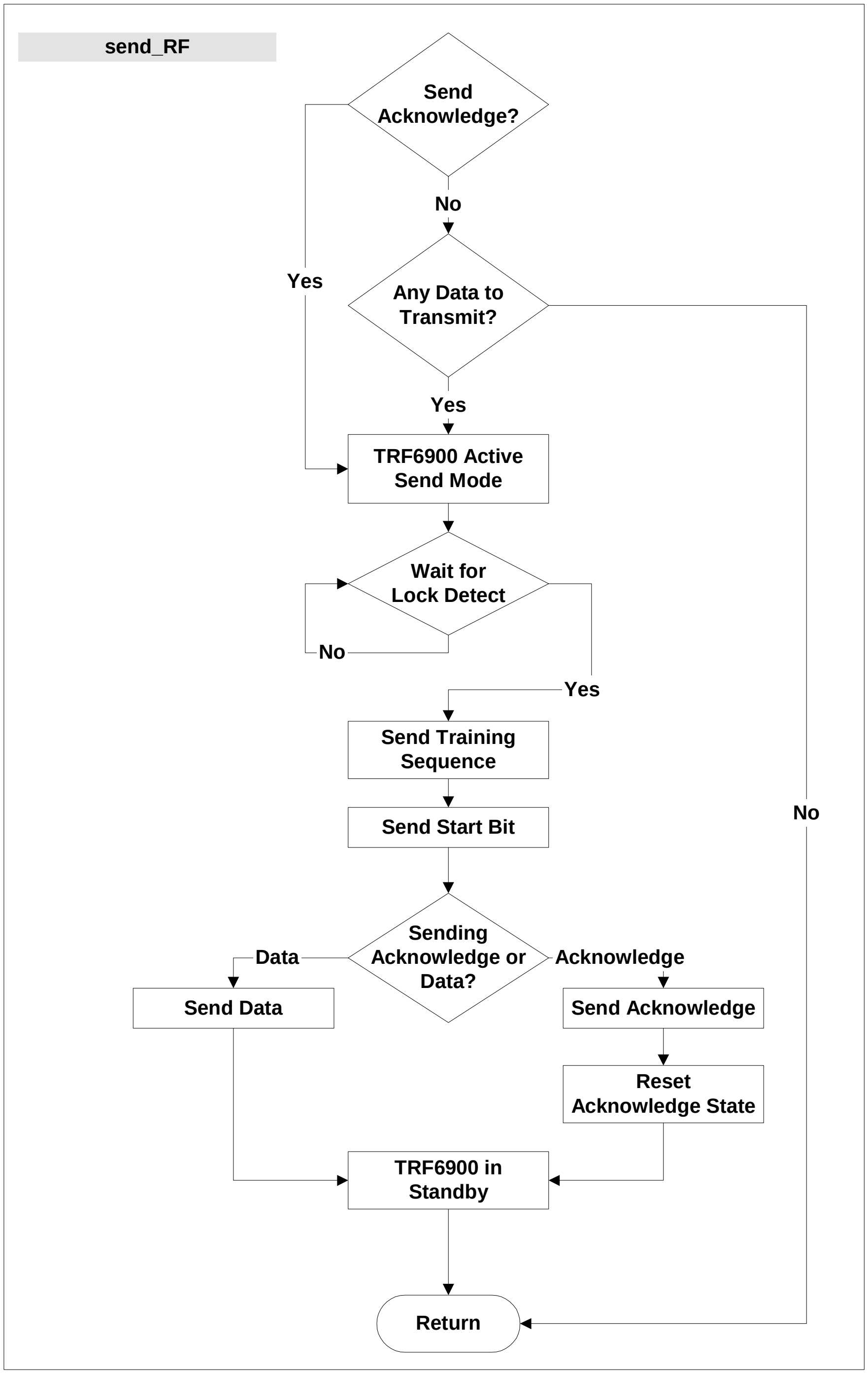


Figure D-1. Logic Diagram for *send_RF*

```
;*************************************** send_RF ************************************************************
; last change: April 3rd 2000
;
; purpose:    sends the received 32 bytes package from RS232 + 2 bytes checksumvia RF as
;             NRZ code, features a shorter training sequence
;             counter value comes from the calling routine
;
;************************************************************************************************************

send_RF

send_RF_acknowledge?
        BIT     #020h,&data_rx_state            ; has an acknowledge to be send?
        JNZ     send_RF_init

data_to_transmit?
        BIT     #04h,&data_rx_state             ; is there any data to send via TRF6900?
        JZ      end_send_RF                     ; no nothing to send

send_RF_init
        DINT

;       CALL    #program_DDS1_send              ; program the DDS_1 register for sending
;       CALL    #program_send_FSK               ; program the mode 1 register for FSK sending
                                                ; obsolete, only necessary if routine used as
                                                ; stand alone
        BIS.B   #stdb_rs232,&P1OUT              ; RS232 driver in standby mode
        BIC.B   #rxd,&P1IE                      ; disable P1.2 Interrupt
        BIS.B   #stdb_trf6900,&P2OUT            ; TRF6900 active, STANDBY(P2.5) is high
        BIS.B   #mode,&P2OUT                    ; Mode 1 is set -> Send mode
        BIC.B   #tx,&P1OUT                      ; TXDATA(P1.4) is reset
        CALL    #wait_lockdet                   ; wait for the lockdetect signal
        MOV     #WDTPW+WDTHOLD,&WDTCTL
                                                ; Password, Watchdog Timer hold
        MOV     0200h(counter),data_r           ; push data to the send register
        MOV     #WDTPW+WDTTMSEL+WDTCNTCL+03h,&WDTCTL
                                                ; Reset, Timer Mode, Password, every 26,04µsec
        BIS.B   #01h,&IE1                       ; enable Watchdog Timer interrupt
        EINT

send_RF_training_sequence                       ; the entire length ca. 1ms, 38 pulses
;       MOV     #03Ch,tr_counter                ; 1.56ms, 60 pulses
        MOV     #026h,tr_counter                ; 1ms, 38 pulses
;       MOV     #09Ah,tr_counter                ; initialize the training sequence counter

send_RF_toggle
        BIS     #CPUOFF+GIE,SR                  ; CPU off
;--------------------------------- Start of the trainings sequence ---------------------------------------------------
        XOR.B   #tx,&P1OUT                      ; toggle TXDATA(P1.4)
        DEC     tr_counter                      ; decrement counter for the training sequence
        JNZ     send_RF_toggle                  ; length of the loop exactly 26,04µsec

send_RF_long_bit
```

```
        BIS     #CPUOFF+GIE,SR                   ; CPU off
;--------------------------------------- Start of the start bit ---------------------------------------------------------
        BIS.B   #tx,&P1OUT                     ; start of the long start-bit 78,12µsec
                                               ; (end by the transmission of the 1st data bit)
        BIS     #CPUOFF+GIE,SR                 ; CPU off
        BIC     #04h,&data_rx_state            ; the RS232 buffer is ready for reception
        BIS     #CPUOFF+GIE,SR                 ; CPU off
        BIS     #CPUOFF+GIE,SR                 ; CPU off
;------------------------------------- End of the start bit -------------------------------------------------------------
        BIC.B   #tx,&P1OUT                     ; reset TXDATA(P1.4)
        BIS     #CPUOFF+GIE,SR                 ; CPU off

send_RF_data
        MOV     #010h,bits_r                   ; init bitcounter, transmit first 16 bits

send_RF_bit_test
        RLC     data_r                         ; push the next data bit to carry
        JC      send_RF_high

send_RF_low
        BIS     #CPUOFF+GIE,SR                 ; CPU off
;--------------------------------------- Start of the Databit -----------------------------------------------------------
        BIC.B   #tx,&P1OUT                     ; reset TXDATA(P1.4)

send_RF_next_word?
        DEC     bits_r                         ; decrement bit counter
        JNZ     send_RF_bit_test
        DECD    counter                        ; decrement word counter
        JZ      send_RF_LED3_on                ; all data has been transmitted
        JN      send_RF_reset_ackn             ; acknowledge has been transmitted
        MOV     0200h(counter),data_r          ; get the next data word
        JMP     send_RF_data                   ; send next word

send_RF_high
        BIS     #CPUOFF+GIE,SR                 ; CPU off
;--------------------------------------- Start of the Data Bit ----------------------------------------------------------
        BIS.B   #tx,&P1OUT                     ; set TXDATA(P1.4)
        JMP     send_RF_next_word?

send_RF_reset_ackn
        BIC     #020h,&data_rx_state           ; reset acknowledge state

send_RF_LED3_on
        BIS     #CPUOFF+GIE,SR                 ; CPU off
        BIS.B   #LED3,&P2OUT                   ; LED3 on

end_send_RF
        MOV     #WDTPW+WDTHOLD,&WDTCTL
                                               ; stop Watchdog Timer
        BIC.B   #stdb_rs232,&P1OUT             ; RS232 driver go active
        BIC.B   #tx,&P1OUT
        BIC.B   #stdb_trf6900,&P2OUT           ; clear STDBY(P2.5), TRF6900 standby mode
        RET
```

Appendix E. *receive_RF*

This routine provides RF reception. It switches the TRF6900 from reception in learn-mode (during receipt of the training sequence) to reception in hold-mode. It is supported by the three interrupt routines, *TA_INT*, *CC2_INT*, and *WDT_INT*.

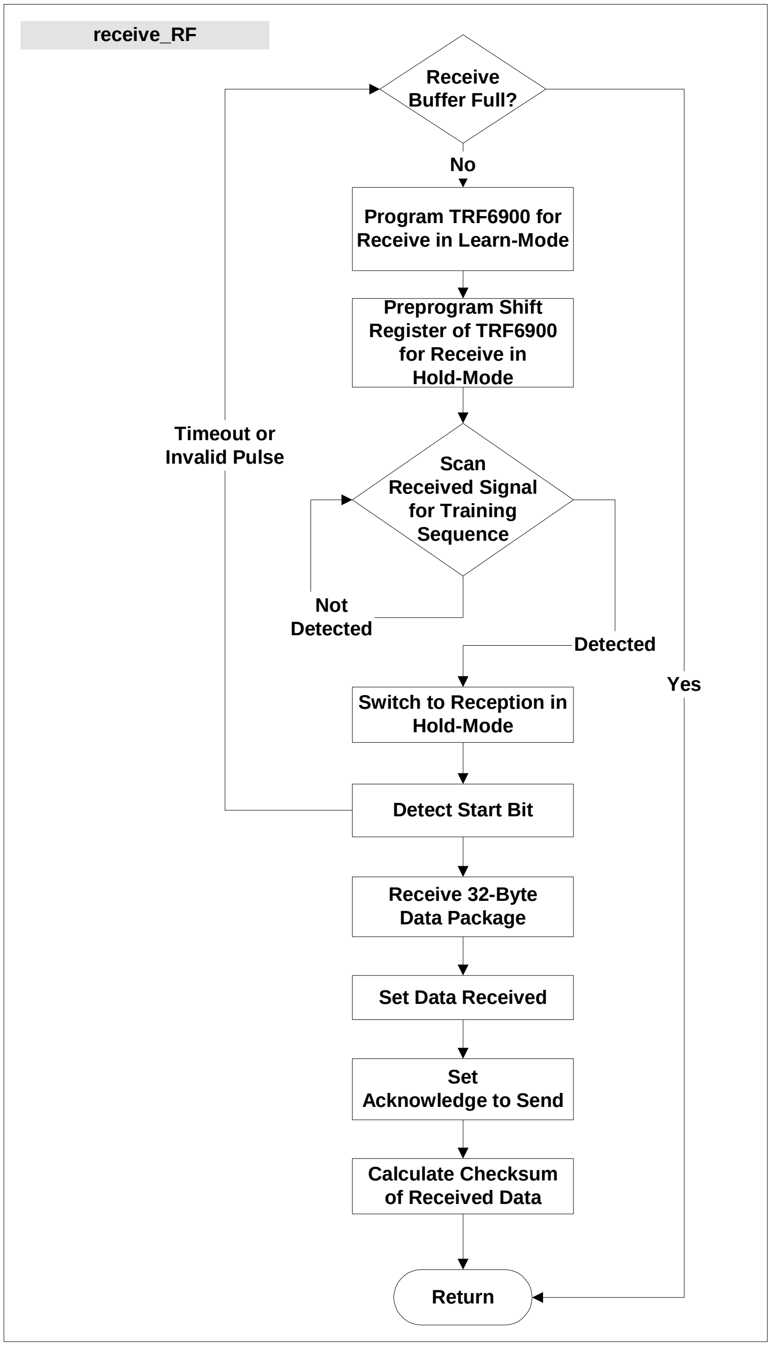

receive_RF
Receive Buffer Full?
No
Program TRF6900 for Receive in Learn-Mode
Preprogram Shift Register of TRF6900 for Receive in Hold-Mode
Timeout or Invalid Pulse
Scan Received Signal for Training Sequence
Not Detected
Detected
Yes
Switch to Reception in Hold-Mode
Detect Start Bit
Receive 32-Byte Data Package
Set Data Received
Set Acknowledge to Send
Calculate Checksum of Received Data
Return

Figure E-1. Logic Diagram for *receive_RF*

```
;************************************** receive_RF **********************************************************
; last change: April 3rd 2000
;
; main routine for code reception via RF
;
; purpose:      receives the 32 bytes data package + 2 bytes checksum and saves it to memory
;               the reception is also supported by several interrupt routines
;               checksum checking implemented (for Flash reprogramming)
;
;***************************************************************************************************************

receive_RF
        BIT     #02h,&data_rx_state             ; is the reception buffer full?
        JNZ     end_receive_RF                  ; yes the data has to be send to desktop first
        CALL    #program_send_FSK               ; program the C-word for reception in
                                                ; learn mode
        MOV     #CCIE+CAP+CMNEG,&CCTL1  ; interrupt enable, capture mode, neg. edge
        CLR     data_r                          ; reset data_r
        CLR     wake_up_counter                 ; reset wake_up_counter
        CLR     RSTAT                           ; reset receive status register, RSTAT = 0,
                                                ; detecting the Trainingssequence
        BIC.B   #mode,&P2OUT                    ; receive FSK in learn mode
        BIS.B   #01h,&IE1                       ; enable Watchdog Timer interrupt
        BIS.B   #stdb_trf6900,&P2OUT            ; TRF6900 active, STANDBY(P2.5) is high
        CALL    #wait_lockdet
        MOV     #TAIE+CLEAR+CONTUP+MCLK,&TACTL
                                                ; interrupt enable, clear Timer_A,
                                                ; continuous up mode, MCLK as clock source
        MOV     #CCIE+CAP+CMANY,&CCTL2  ; interrupt enable, capture mode, both edges
        BIS     #08h,&data_rx_state             ; for suppressing strobe pulse at the end
                                                ; of the programming routine
        BIS.B   #LED3,&P2OUT                    ; switch on the system mode LED
        CALL    #program_receive_FSK_hold       ; goto hold mode
        BIC     #08h,&data_rx_state             ; reset the the "suppressing mode"
        BIC.B   #data,&P1OUT                    ; reset data

loop_receive_training_seq
;------------------------ scanning the received signal for the training sequence ------------------------------
        CMP     #08h,wake_up_counter            ; 8 equal pulses in succession
        JL      loop_receive_training_seq       ; no the result is less than 8 equal pulses
                                                ; in a row
        BIC     #022h,&TACTL                    ; stop Timer_A and disable interrupt
        BIC     #CCIE,&CCTL2                    ; disable interrupt
        INCD    RSTAT                           ; RSTAT = 4

start_bit_reception                             ; waiting for the start_bit
        BIS.B   #strobe,&P1OUT                  ; set strobe(P1.5)
        BIC.B   #strobe,&P1OUT                  ; clear strobe(P1.5)
        MOV     #TAIE+CLEAR+CONTUP+MCLK,&TACTL
                                                ; interrupt enable, clear Timer_A, continuous
                                                ; up mode, MCLK as clock source
        MOV     #CCIE+CAP+CMANY,&CCTL2  ; interrupt enable, capture mode, both edges

loop_start_bit
```

```
        CMP     #04h,RSTAT                      ; has the start bit been detected?
        JEQ     loop_start_bit                  ; wait for the start bit
        JN      receive_RF                      ; the received sequence is invalid
        BIC.B   #rxd,&P1IE                      ; disable P1.2 interrupt
;-------------------------------------------- start bit detected -------------------------------------------------------

init_data_reception                             ; RSTAT = 6, Start Bit detected, Data Reception
        NOP                                     ; insert for the right timing
        NOP
        MOV     #WDTPW+WDTTMSEL+WDTCNTCL+03h,&WDTCTL
                                                ; Reset, Timer Mode, Password, every 26µs
        MOV     #022h,counter                   ; Initialize counter

init_rx_bit_counter
        CLR     bits_r                          ; Reset bitcounter

word_reception_loop
        BIS     #CPUOFF+GIE,SR                  ; go to sleep!
                                                ; wake up from WDT (Timer Mode, every 26µs)
;       NOP                                     ; insert for exact timing
;       NOP                                     ;
;       NOP                                     ;
;       NOP                                     ;
;       NOP                                     ;
;       NOP
;       NOP
;       NOP
;       XOR.B   #LED3,&P2OUT                    ; for test purpose
        BIT.B   #08h,&P1IN                      ; is RXDATA high or low?

read_data
        RLC     data_r                          ; push carry into the data register
        INC     bits_r
        CMP     #010h,bits_r                    ; receive 16 bits in row
        JNE     word_reception_loop             ; haven't received 8bits yet

store_data
        INV     data_r                          ; the received data is inverted!
        MOV     data_r,0222h(counter)           ; store received data to RAM
        DECD    counter                         ; next storage register
        JNZ     init_rx_bit_counter             ; receive the next word
        BIC.B   #LED3,&P2OUT                    ; system LED off for test purpose
        BIS     #02h,&data_rx_state             ; 2 stands for data received,
                                                ; has to be send to desktop via RS232
        BIS     #020h,&data_rx_state            ; initialize the acknowledge state

end_receive_RF
        BIC     #CCIE,&CCTL2                    ; disable CCR2 interrupt
        BIC     #022h,&TACTL                    ; stop Timer_A and disable interrupt
        MOV     #WDTPW+WDTHOLD,&WDTCTL
                                                ; stop Watchdog Timer
        CALL    #checksum_r                     ; calculate checksum of received data
        BIC.B   #stdb_trf6900,&P2OUT            ; clear STDBY(P2.5), TRF6900 in standby mode
        RET
```

Appendix F. *rs232_send*

If the checksum is correct, *rs232_send* transmits the received data to the connected PC. Because the routines for RS232 communication do not affect the RF link, they are documented only within the code listing.

```
;*************************************** rs232_send *******************************************************
; last change: April 12th 2000
;
; purpose:     the transmission of the received data from TRF6900 to the PC via RS232-Port
;
;**********************************************************************************************************

rs232_send
        BIT      #02h,&data_rx_state            ; is there any data in the reception buffer,
                                                ; which hasn't been sent to the desktop?
        JZ       end_rs232_send                 ; no the reception buffer is empty
        BIC.B    #cts,&P2OUT                    ; reset CTS
        BIC.B    #stdb_rs232,&P1OUT             ; activate RS232 driver
        MOV      #032Eh,wait_r                  ; initialize wait register for ca. 1ms
                                                ; waiting loop
        CALL     #wait_x_cycles                 ; wait for RS232 driver ready to transmit
        MOV.B    #XTS+DIVA_2,&BCSCTL1           ; set the divider for ACLK to 2,
                                                ; high frequency oscillator
        MOV      #020h,counter                  ; Initialize counter
        BIS.B    #01h,&IE1                      ; enable Watchdog Timer interrupt
        MOV      #WDTPW+WDTTMSEL+WDTCNTCL+07h,&WDTCTL
                                                ; Reset, Timer Mode, Password, every 52.08µs
        EINT                                    ; general Interrupt enable

rs232_send_init
        MOV.B    0223h(counter),data_r          ; move the first received word into the output
                                                ; buffer
        MOV      #0Ah,bits_r                    ; send only 8bit in a row
        BIS      #0100h,data_r                  ; prepare stop bit
        CLRC                                    ; set carry, prepare start bit
        RLC      data_r                         ; prepare the output buffer for data
                                                ; transmission

rs232_send_loop
        RRC      data_r                         ; push next bit to carry for transmission
        JNC      rs232_send_low

rs232_send_high
        BIS      #CPUOFF+GIE,SR                 ; CPU off
        BIS.B    #txd,&P1OUT                    ; set TXD
        DEC      bits_r                         ; decrement bit counter
        JNZ      rs232_send_loop

rs232_send_next_byte
        DEC      counter                        ; decrement byte counter
        JNZ     rs232_send_init                 ; get the next byte for transmission
        BIC     #02h,&data_rx_state             ; the buffer is ready to receive from TRF6900
                                                ; the next data
        JMP     end_rs232_send_lp               ;

rs232_send_low
        BIS     #CPUOFF+GIE,SR                  ; CPU off
        BIC.B   #txd,&P1OUT                     ; reset TXD
        DEC     bits_r                          ; decrement bit counter
          JNZ     rs232_send_loop
          JMP     rs232_send_next_byte

end_rs232_send_lp                               ; generate the last change of the TXD
          BIS     #CPUOFF+GIE,SR                ; CPU off
          BIS.B   #txd,&P1OUT                   ; TXD by default high, data toggles the TXD

end_rs232_send
          DINT                                  ;
          MOV     #WDTPW+WDTHOLD,&WDTCTL
                                                ; stop Watchdog Timer
          BIS.B   #LED3,&P2OUT                  ; set System LED for test purpose
          RET
```

Appendix G. I^{2}C Routines

These routines have been written to drive an I^{2}C LCD, for display of received data using an additional board connected to the 14-pin connector of the EVK. The LCD is driven by a chip-on-glass driver (Type PCF2119x). These routines are documented only within the code listing. The schematics and Gerber files for this additional board are available on the MSP430 CDROM.

```
;******************************************* IIC_start *******************************************************
; last change: January 28th 2000
;
; generates the start condition for IIC
;
;*************************************************************************************************************

IIC_start
      BIS.B   #SDA,&P1OUT                  ; set SDA of IIC-Bus
      BIS.B   #SCL,&P1OUT                  ; set SCL of IIC-Bus
      BIC.B   #SDA,&P1OUT                  ; start condition for the IIC-Bus
      BIC.B   #SCL,&P1OUT                  ; reset CLK

end_IIC_start
      RET

;****************************************** IIC_stop ********************************************************
; last change: January 28th 2000
;
; generates a stop condition for IIC
;
;*************************************************************************************************************

IIC_stop
      BIS.B   #SCL,&P1OUT                   ; set SCL
      BIS.B   #SDA,&P1OUT                   ; stop condition for the II_C

end_IIC_stop
      RET

;************************************** IIC_address_LCD *************************************************
; last change: January 28th 2000
;
; addresses the LCD driver Philips PCF2119x
;
;*************************************************************************************************************

IIC_address_LCD
        MOV    #07600h,data_r
        CALL   #IIC_send_loop_ini

end_IIC_address_LCD
        RET
```

```
;****************************************** IIC_internal_voltage ********************************************
; last change: January 28th 2000
;
; initializes the internal voltage generator for the LCD
; sets the Va value to ca. 5V
;
;****************************************************************************************************************

IIC_internal_voltage
        CALL    #IIC_start
          CALL    #IIC_address_LCD

;IIC_control_set
          CLR       data_r                              ; control byte for function settings no r/w
          CALL    #IIC_send_loop_ini

;IIC_function_set
          MOV     #03500h,data_r                      ; DL=8bits, M=2 line by 16 display,
                                                        ; SL=MUX 1:18, H=extended instruction set
          CALL    #IIC_send_loop_ini

IIC_program_voltage
          MOV     #09B00h,data_r                      ; binary voltage value for Va=22h >> ca. 5V
          CALL    #IIC_send_loop_ini
          CALL    #IIC_stop

end_IIC_internal_voltage
          RET

;***************************************** IIC_init_EVK_string *********************************************
; last change: February 28th 2000
;
; writes the string "MSP EVKTRF6900  RF link" to the IIC-RAM space (2DF-2C0)
;
;****************************************************************************************************************

IIC_init_EVK_string
        MOV     #04D53h,&0242h                          ; MS
        MOV     #0502Dh,&0240h                          ; P
        MOV     #04556h,&023Eh                          ; EV
        MOV     #04B54h,&023Ch                          ; KT
        MOV     #05246h,&023Ah                          ; RF
        MOV     #03639h,&0238h                          ; 69
        MOV     #03030h,&0236h                          ; 00
        MOV     #02020h,&0234h                          ;
        MOV     #05246h,&0232h                          ; RF
        MOV     #02D4Ch,&0230h                          ; -L
        MOV     #0696Eh,&022Eh                          ; in
        MOV     #06B20h,&022Ch                          ; k
        MOV     #04B69h,&022Ah                          ; Ki
        MOV     #0636Bh,&0228h                          ; ck
        MOV     #07374h,&0226h                          ; st
        MOV     #03236h,&0224h                          ; 26
```

```
end_IIC_init_EVK_string
        RET

;****************************************** IIC_initialize_LCD ***********************************************
; last change: January 28th 2000
;
; configures the LCD-driver for the display
;
;*************************************************************************************************************

IIC_initialize_LCD
        CALL    #IIC_start
        CALL    #IIC_address_LCD

IIC_control_set
        CLR     data_r
        CALL    #IIC_send_loop_ini

IIC_function_set
        MOV     #03400h,data_r                  ; DL=8bits, M=2line by 16 display,
                                                ; SL=MUX 1:18, H=use basic instruction set
        CALL    #IIC_send_loop_ini

IIC_display_on
        MOV     #0F00h,data_r                   ; D=display on , C=cursor on,
                                                ; B=cursor character blink on
        CALL    #IIC_send_loop_ini

;IIC_entry_mode
;       MOV     #0600h,data_r                   ; I/D=increment, S=display freeze
;       CALL    #IIC_send_loop_ini

;IIC_cursor_display_shift
;       MOV     #01400h,data_r                  ; S/C=display shift, R/L=right shift
;       CALL    #IIC_send_loop_ini

IIC_DDRAM_start
        MOV     #08000h,data_r
        CALL    #IIC_send_loop_ini

        CALL    #IIC_stop

end_IIC_initialize_LCD
        RET


;**************************************** IIC_LCD_display **************************************************
; last change: February 28th 2000
;
; displays the received data on the LCD (BT21605)
;
;*************************************************************************************************************

IIC_LCD_display
```

```
        MOV     #010h,counter                ; for IIC display loop
        CALL    #IIC_start                   ; send start condition
        CALL    #IIC_address_LCD             ; send address of the LCD driver

;IIC_control_set
        MOV     #08000h,data_r               ; send control byte for function set,
                                             ; first control byte
        CALL    #IIC_send_loop_ini

IIC_LCD_home                                 ; sets DDRAM to 0
        MOV      #0200h,data_r               ; return home
        CALL     #IIC_send_loop_ini

IIC_write_text
        MOV      #04000h,data_r              ; control byte, write data
        CALL     #IIC_send_loop_ini

IIC_LCD_display_loop_1                       ; 1. display line
        MOV.B    0233h(counter),data_r       ; push the next character to the buffer
        XOR.B    #080h,data_r                ; convert to IIC-LCD format
        SWPB     data_r
        CALL     #IIC_send_loop_ini
        DEC      counter
        JNZ      IIC_LCD_display_loop_1

        CALL     #IIC_stop                   ; send stop condition

        MOV      #010h,counter               ; for IIC display loop
        CALL     #IIC_start                  ; send start condition
        CALL     #IIC_address_LCD            ; send address of the LCD driver

;IIC_control_set
        MOV      #08000h,data_r              ; send control byte for function set,
                                             ; first control byte
        CALL     #IIC_send_loop_ini

IIC_second_line
        MOV      #0C000h,data_r              ; set DDRAM 40h
        CALL     #IIC_send_loop_ini

;IIC_write_text
        MOV      #04000h,data_r              ; send control byte for writing data
        CALL     #IIC_send_loop_ini

IIC_LCD_display_loop_2
        MOV.B    0223h(counter),data_r       ; move received byte into data_r
        XOR.B    #080h,data_r                ; convert to IIC-LCD format
        SWPB     data_r
        CALL     #IIC_send_loop_ini
        DEC      counter
        JNZ      IIC_LCD_display_loop_2

        CALL     #IIC_stop
```

```
end_IIC_LCD_display
        RET



;****************************************** IIC_send_loop **************************************************
; last change: January 27th 2000
;
; purpose: sends 8 bits via IIC-Bus
;
;************************************************************************************************************

IIC_send_loop_ini
        MOV     #08h,bits_r                     ; bit counter for IIC loop

IIC_send_loop
        RLC     data_r                          ; push the first bit to carry
        JC      IIC_high                        ; if the bit's high

IIC_low
        BIC.B   #SDA,&P1OUT                     ; reset SDA
        NOP
        NOP

IIC_clk
        BIS.B   #SCL,&P1OUT                     ; set IIC clock
        NOP
        NOP
        BIC.B   #SCL,&P1OUT                     ; reset IIC clock
        DEC     bits_r                          ; decrement bit counter
        JNZ     IIC_send_loop

IIC_ackn
        BIS.B   #SDA,&P1OUT                     ; acknowledge phase
        NOP
        NOP
        BIS.B   #SCL,&P1OUT                     ; generate clock pulse
        NOP
        NOP
        BIC.B   #SCL,&P1OUT                     ;

end_IIC_send_loop
        RET

IIC_high
        BIS.B   #SDA,&P1OUT                     ; set SDA
        NOP
        JMP     IIC_clk
```

Appendix H. *checksum_s*

The purpose of this routine is to calculate the checksum of the data package received from RS232, which is used to enable an acknowledge from the receiver. The transmitter adds the calculated checksum to the actual data package and transmits them together. The receiver compares the checksum it calculates with the checksum it received with the data package. If the checksums are equal, an acknowledge is sent to the transmitter.

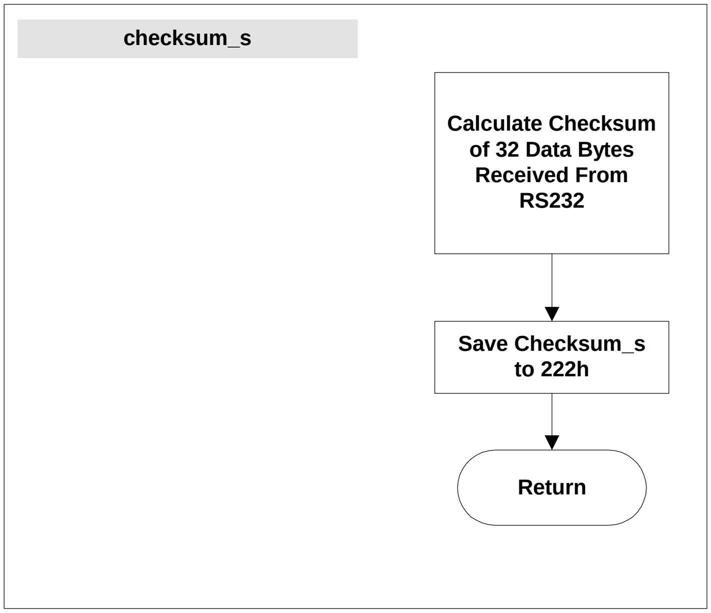


Figure H-1. Logic Diagram for *checksum_s*

```
;**************************************** checksum_s ********************************************************
; last change: April 11th 2000
;
; purpose:  calculate the checksum of the received data package from RS232, for secure data
;           transmission
;           (Flash reprogramming via RF)
;
;*****************************************************************************************************************

checksum_s
        MOV     #020h,counter                   ; initialize counter
        CLR     chcksum_s                       ; reset checksum

checksum_s_loop
        ADD     0200h(counter),chcksum_s        ; calculate checksum of the whole data
                                                ; package, that has to be send
        DECD    counter
        JNZ     checksum_s_loop
        MOV     chcksum_s,&0222h                ; save checksum

end_checksum_s
        RET
```

Appendix I. *checksum_r*

This routine calculates the checksum of the data package received from RF for comparison with the checksum received with the data package. If the two checksums are equal, an acknowledge is send to the transmitter.

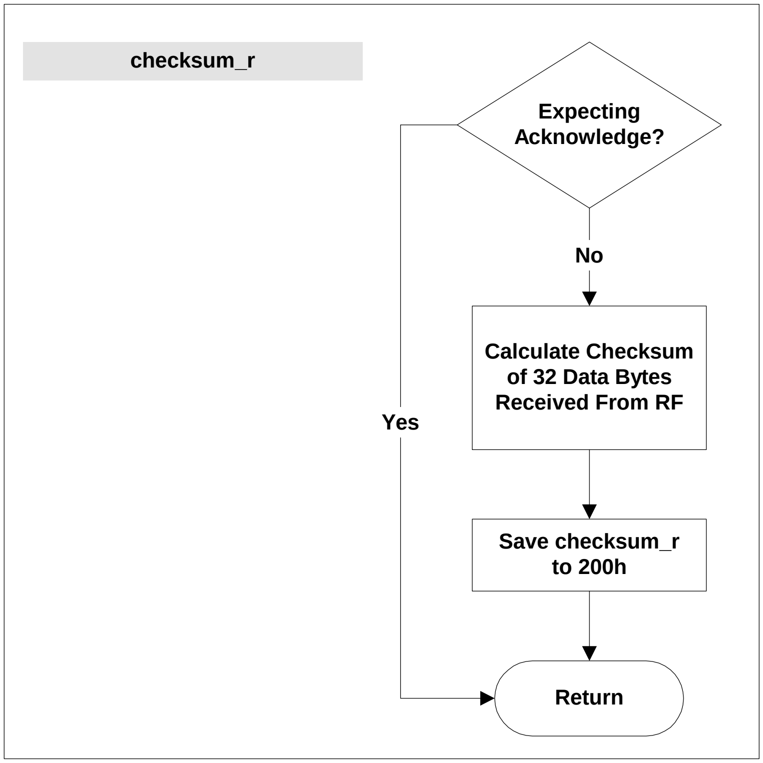


Figure I-1. Logic Diagram for *checksum_r*

```
;**************************************** checksum_r *******************************************************
; last change: April 11th 2000
;
; purpose:    the checksum of the received data package from RF, for reliable data transmission
;             (Flash reprogramming via RF)
; ;**************************************************************************************************************
checksum_r
        BIT     #040h,&data_rx_state        ; expecting acknowledge?
        JNZ     end_checksum_r              ; no need to calculate the checksum
        MOV     #020h,counter               ; initialize counter
        CLR     chcksum_r                   ; reset checksum

checksum_r_loop
        ADD     0222h(counter),chcksum_r    ; calculate checksum of the entire data to send
        DECD    counter
        JNZ     checksum_r_loop
        MOV     chcksum_r,&0200h            ; save checksum

end_checksum_r
        RET
```

Appendix J. *check_checksum_s*

On the transmitter side, this routine compares the checksum of the transmitted data package via RF, with the checksum received from the receiver as acknowledge. If the two checksums are equal, an acknowledge is sent to the connected PC.

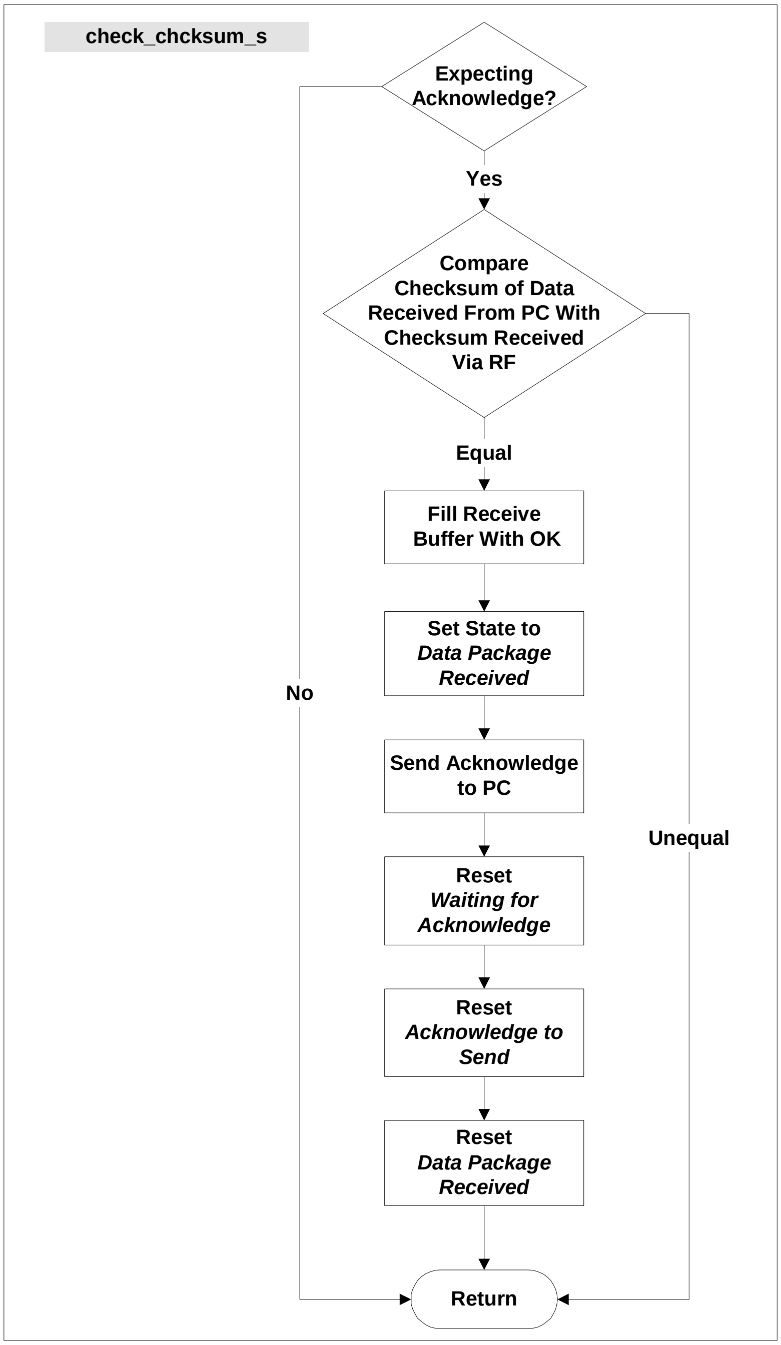

check_chcksum_s
Expecting Acknowledge?
Yes
No
Compare Checksum of Data Received From PC With Checksum Received Via RF
Equal
Unequal
Fill Receive Buffer With OK
Set State to *Data Package Received*
Send Acknowledge to PC
Reset *Waiting for Acknowledge*
Reset *Acknowledge to Send*
Reset *Data Package Received*
Return

Figure J-1. Logic Diagram for *check_checksum_s*

```
;********************************** check_checksum_s ************************************************
; last change: February 29th 2000
;
; purpose:    compares the received checksum from the receiver, with the checksum of the
;             received data
;             from the PC via RS232
;
;****************************************************************************************************
check_checksum_s                              ;
        BIT     #040h,&data_rx_state          ; check if waiting for an acknowledge
        JZ      end_check_checksum_s          ;
        CMP     &0222h,&0244h                 ; compare the both checksums
        JNE     end_check_checksum_s          ; no the received data isn't valid

check_checksum_s_ok                           ;
        MOV     #06F6Bh,&0242h                ; move "ok" to the first word
        MOV     #06F6Bh,&0240h                ;
        MOV     #06F6Bh,&023Eh                ;
        MOV     #06F6Bh,&023Ch                ;
        MOV     #06F6Bh,&023Ah
        MOV     #06F6Bh,&0238h
        MOV     #06F6Bh,&0236h
        MOV     #06F6Bh,&0234h
        MOV     #06F6Bh,&0232h
        MOV     #06F6Bh,&0230h
        MOV     #06F6Bh,&022Eh
        MOV     #06F6Bh,&022Ch
        MOV     #06F6Bh,&022Ah
        MOV     #06F6Bh,&0228h
        MOV     #06F6Bh,&0226h
        MOV     #06F6Bh,&0224h

        BIS     #02h,&data_rx_state           ; set the data package received state
        CALL    #rs232_send                   ; send the "ok" to the PC
        BIC     #040h,&data_rx_state          ; reset the state "waiting for acknowledge"
        BIC     #020h,&data_rx_state          ; reset the state "acknowledge to send"
        BIC     #02h,&data_rx_state           ; reset the state "data_package_received"
end_check_checksum_s                          ;
        RET
```

Appendix K. *check_checksum_r*

This receiver-side routine compares the checksum received with the data package to the checksum calculated by the receiver. If the checksums are equal, the data are sent to the connected PC, and an acknowledge is sent to the transmitter.

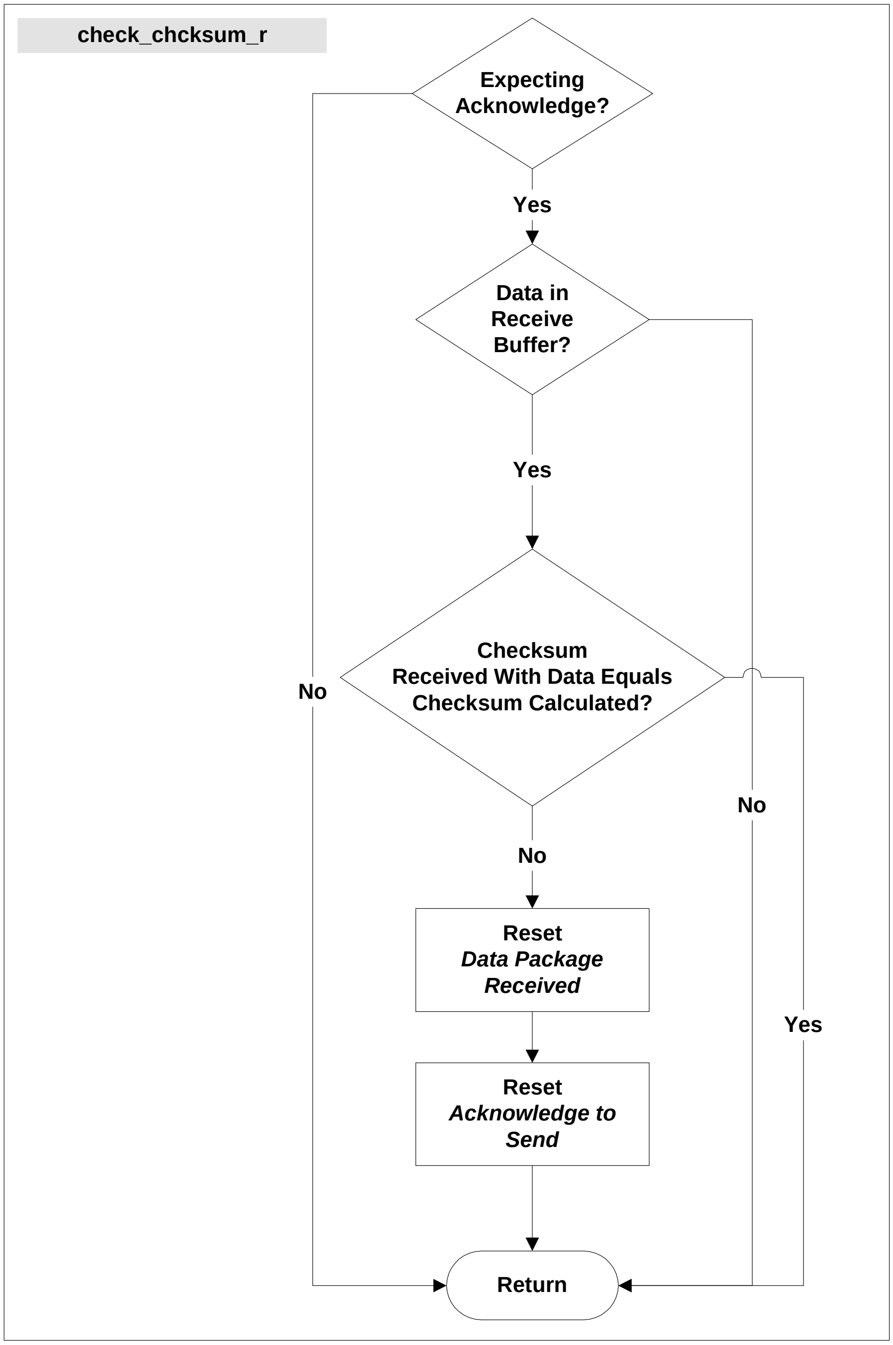

check_chcksum_r
Expecting Acknowledge?
Yes
Data in Receive Buffer?
Yes
Checksum Received With Data Equals Checksum Calculated?
No
No
No
Yes
Reset *Data Package Received*
Reset *Acknowledge to Send*
Return

Figure K-1. Logic Diagram for *check_checksum_r*

```
;********************************* check_checksum_r ****************************************************
; last change: April 13th 2000
;
; purpose:    - Compares the received checksum, that has been calculated by the transmitter for
;               the received data package via RS232, with the checksum calculated on the receiver ;
  side of the received data package via RF.
;               - If the two checksums are equal, the received data package is send of via RS232,
;               and the checksum is send as an acknowledge back to the sender.
;               - IF the checksums aren't equal the package is not send via RS232 to the PC, and
;               no acknowledge will be send to the sender.
;*************************************************************************************************************
check_checksum_r
          BIT    #040h,&data_rx_state             ; check if waiting for an acknowledge
          JNZ    end_check_checksum_r
          BIT    #02h,&data_rx_state              ; is there any data in the reception buffer,
                                                  ; which hasn't been sent to the desktop?
          JZ     end_check_checksum_r             ; no, no new data package
          CMP    &0200h,&0244h                    ; are the checksums equal?
          JEQ    end_check_checksum_r             ; yes the data is valid

check_checksum_r_inv
          BIC    #02h,&data_rx_state              ; reset the data package received state
          BIC    #020h,&data_rx_state             ; reset the acknowledge to send state

end_check_checksum_r
          RET
```

Appendix L. Wait Loops

This appendix documents the wait routines used by the implemented RF link. The simplicity of these routines allows them to be documented only within the code.

```
;***************************************** wait loops *****************************************************
; last change: August 11th 1999
;
; purpose:  used for various timings, e.g. NRZ reception and transmission
;
;***********************************************************************************************************

wait_x_cycles                         ; waits 13 + x times 3 cycles
        DEC     wait_r
        JNZ     wait_x_cycles
        RET

end_wait_x_cycles


;*************************************** wait for lockdetect ***********************************************
; last change: September 6th 1999
;
; wait routine for the Lockdetect signal
;
;***********************************************************************************************************

wait_lockdet
        BIT.B   #lockdet,&P2IN        ; is the LOCKDET(P2.4) set?
        JZ      wait_lockdet          ; not yet

end_wait_lockdet
        RET
```

Appendix M. *TA_INT* — Timer_A Interrupt Routine

This routine handles the interrupts from Timer_A and determines which dedicated interrupt routine should be addressed.

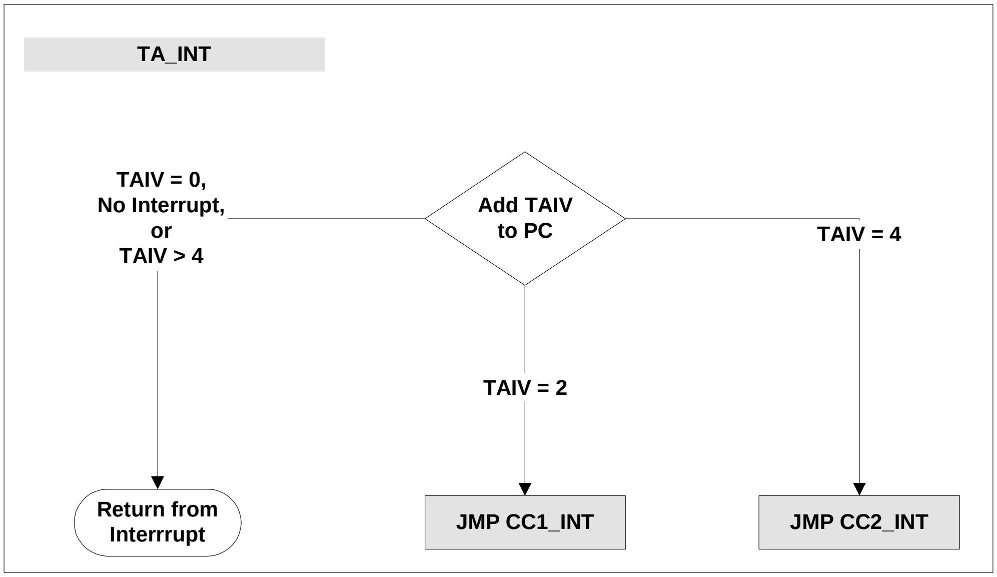


Figure M-1. Logic Diagram for *TA_INT*

```
;************************************** Timer_A Interrupt routine *****************************************
; last change: July 27th 1999
;
; purpose:    handle the Timer_A interrupts, and decide which dedicated routine should be
;             addressed. (CC1_INT / RS232 reception, CC2_INT / RF reception)
;**************************************************************************************************************

TA_INT
        ADD     &TAIV,PC
        RETI
        JMP     CC1_INT                       ; RS232 reception -> falling edge of the
                                              ; start bit
        JMP     CC2_INT                       ; RF reception -> every edge of
                                              ; the rx-signal
        RETI
        RETI
        RETI
```

Appendix N. *CC1_INT* — Capture Compare 1 Interrupt Routine

This routine is used to set up a software UART. The terminal program *TRF6900* Demo 1.16 transmits data to the EVM board via RS232—*CC1_INT* handles data reception from *TRF6900*.

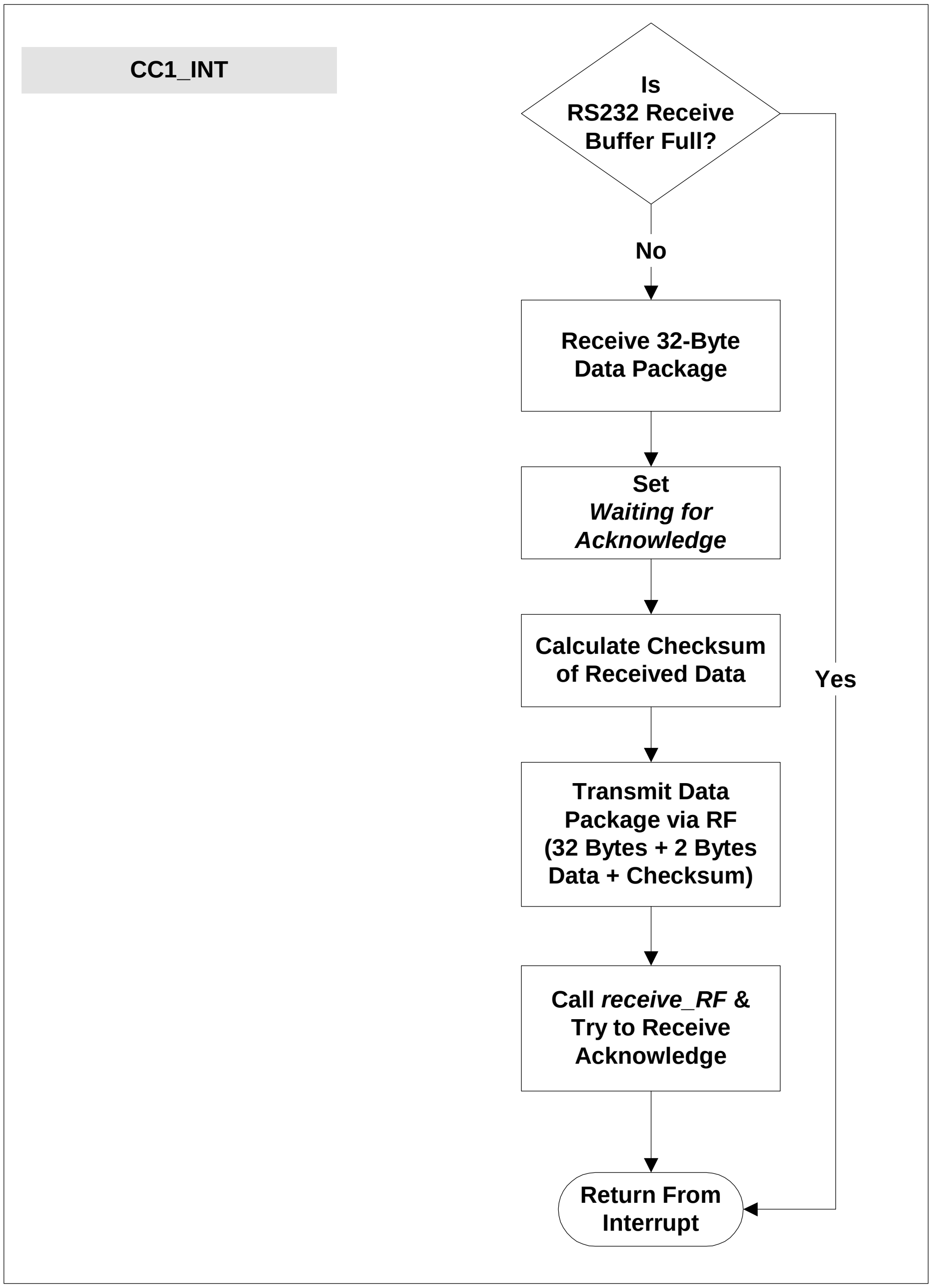

CC1_INT
Is RS232 Receive Buffer Full?
No
Yes
Receive 32-Byte Data Package
Set *Waiting for Acknowledge*
Calculate Checksum of Received Data
Transmit Data Package via RF (32 Bytes + 2 Bytes Data + Checksum)
Call *receive_RF* & Try to Receive Acknowledge
Return From Interrupt

Figure N-1. Logic Diagram for *CC1_INT*

```
;****************************** Capture Compare 1 Register *******************************************
; last change: April 11th 2000
;
; purpose:  RS232-Reception
;
;***********************************************************************************************************

CC1_INT
      BIC     #CCIE,&CCTL1                  ; disable CCR1 interrupt
      BIT     #04h,&data_rx_state           ; is the RS232 buffer full?
      JNZ     end_CC1_INT                   ; yes, do not receive further data
      MOV     #08h,bits_r                   ; init bit counter
      MOV     #020h,counter                 ; init byte counter
      BIC     #022h,&TACTL                  ; stop Timer_A and disable interrupt
      BIC     #CCIE,&CCTL2                  ; disable interrupt
      EINT

rs232_init_WDT
      MOV.B #XTS+DIVA_2,&BCSCTL1            ; set the divider for ACLK to 2,
                                            ; high frequency oscillator
      MOV     #WDTPW+WDTTMSEL+WDTCNTCL+07h,&WDTCTL
                                            ; Reset, Timer Mode, Password, every 52.08µs

rs232_rec_data
      BIS     #CPUOFF+GIE,SR                ; CPU off
      BIC.B   #LED3,&P2OUT                  ; LED3 off
      BIT.B   #rxd,&P1IN                    ; is the rx line high, or low?

rs232_push_buffer
      RRC     data_r                        ; push carry to reception buffer
      DEC     bits_r                        ; decrement bit counter
      JNZ     rs232_rec_data                ; read the next bit

rs232_stop_bit
      BIS     #CPUOFF+GIE,SR                ; CPU off
      BIS     #04h,&data_rx_state           ; set the data reception state to received!
      MOV     #WDTPW+WDTHOLD,&WDTCTL
                                            ; stop watchdog timer

rs232_start_bit                             ; waits for the next start bit
      BIT.B   #rxd,&P1IN                    ; wait for the falling edge of the
                                            ; start bit for synchronization
      JNZ     rs232_start_bit
      MOV     #WDTPW+WDTTMSEL+WDTCNTCL+07h,&WDTCTL
                                            ; Reset, Timer Mode, Password, every 52.08µs
      MOV     #08h,wait_r                   ; init wait parameter
      CALL    #wait_x_cycles
      MOV     #08h,bits_r                   ; init bit counter
      SWPB    data_r                        ; swap high and low byte of the receive buffer
      MOV.B data_r,0201h(counter)           ; store the received data to RAM
                                            ; for RF-Transmission
      DEC     counter                       ; decrement the byte counter
      JNZ     rs232_rec_data                ; get the next byte
```

```
end_CC1_INT
        MOV     #WDTPW+WDTHOLD,&WDTCTL
                                        ; stop watchdog timer
        BIS     #040h,&data_rx_state    ; set the status on waiting for acknowledge
        CALL    #checksum_s             ; built checksum over received data
        MOV     #022h,counter           ; init counter
        CALL    #send_RF                ; send the received data out
        BIC     #CPUOFF+GIE,0(SP)       ; wake up from sleep mode
        MOV     #receive_RF,2(SP)       ; do not return to the original address
                                        ; before the interrupt request, but start
                                        ; at the beginning of the receive_RF
                                        ; subroutine
        RETI
```

Appendix O. *CC2_INT* — Capture Compare 2 Interrupt Routine

This routine detects the training sequence and the start bit of the transmitted data package. It is active during execution of *receive_RF*. Reception in several stages; this interrupt routine handles switching from one stage to another.

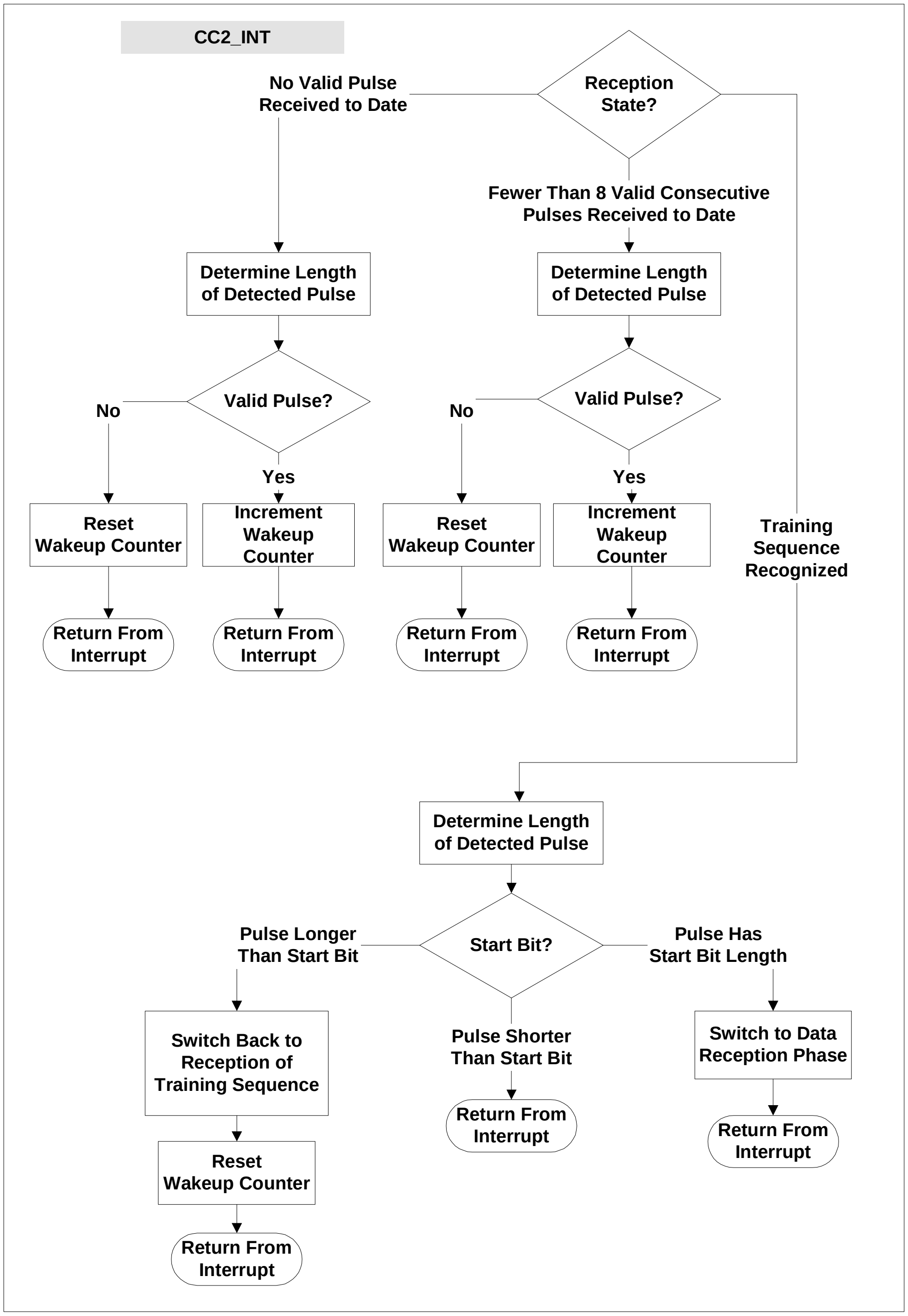

CC2_INT
Reception State?
No Valid Pulse Received to Date
Fewer Than 8 Valid Consecutive Pulses Received to Date
Training Sequence Recognized
Determine Length of Detected Pulse
Valid Pulse?
No
Yes
Reset Wakeup Counter
Increment Wakeup Counter
Return From Interrupt
Determine Length of Detected Pulse
Valid Pulse?
No
Yes
Reset Wakeup Counter
Increment Wakeup Counter
Return From Interrupt
Return From Interrupt
Return From Interrupt
Determine Length of Detected Pulse
Start Bit?
Pulse Longer Than Start Bit
Pulse Shorter Than Start Bit
Pulse Has Start Bit Length
Switch Back to Reception of Training Sequence
Reset Wakeup Counter
Return From Interrupt
Return From Interrupt
Switch to Data Reception Phase
Return From Interrupt

Figure O-1. Logic Diagram for *CC2_INT*

```
;******************************* Capture Compare 2 Register ********************************************
; last change: October 19th 1999
;
; purpose:  RF-Reception
;
;************************************************************************************************************

CC2_INT                                 ; supports receive_RF
        MOV     #WDTPW+WDT26MS+WDTCNTCL+WDTTMSEL,&WDTCTL
                                        ; ACLK, ca.106ms, Timer Mode, Reset
        MOV     RRFTAB(RSTAT),PC        ; conditional jump depends on RSTAT
                                        ; RSTAT = 0, detecting the Trainingsequence
                                        ; RSTAT = 1, Trainingsquence detected, waiting
                                        ;               for the Start Bit
                                        ; RSTAT = 2, Start Bit detected, Data Reception
RRFTAB     DW   RSTAT00
           DW   RSTAT01
           DW   RSTAT10

RSTAT00
        MOV     &CCR2,res_new_r         ; save Reference Capture value
        MOV     res_new_r,res_r         ; copy Timer_A value
        SUB     res_old_r,res_r         ; subtract the current Timer_A value from the
                                        ; old one -> Bitwidth in cycles in res_r
        MOV     res_new_r,res_old_r     ; current value now -> old value later

test_res_r00
        SUB     #038h,res_r             ; subtract the average value from the
                                        ; measured value
        CMP     #010h,res_r             ; is the detected signal 51-63 cycles long?
        JHS     no_valid_pulse
        INCD    RSTAT                   ; first valid pulse detected
        INC     wake_up_counter         ; count this valid pulse
        RETI

no_valid_pulse
        CLR     RSTAT                   ; no the signal doesn't fit the wakeup sequence
        CLR     wake_up_counter         ; reset the wake_up_counter, received an
                                        ; invalid pulse
        RETI

RSTAT01
        MOV     &CCR2,res_new_r         ; save Reference Capture value
        MOV     res_new_r,res_r         ; copy Timer_A value
        SUB     res_old_r,res_r         ; subtract the current Timer_A value from the
                                        ; old one -> Bitwidth in cycles in res_r
        MOV     res_new_r,res_old_r     ; current value now -> old value later

test_res_r01
        SUB     #038h,res_r             ; subtract the average value from the
                                        ; measured value
        CMP     #010h,res_r             ; is the detected signal 51-63 cycles long?
        JHS     no_valid_pulse
        INC     wake_up_counter         ; next valid pulse
        RETI

RSTAT10
        MOV     &CCR2,res_new_r           ; save Reference Capture value
        MOV     res_new_r,res_r           ; copy Timer_A value
        SUB     res_old_r,res_r           ; subtract the current Timer_A value from
                                          ; the old one -> Bitwidth in cycles in res_r
        MOV     res_new_r,res_old_r       ; current value now -> old value later

test_res_r10
        SUB     #0ACh,res_r               ; subtract the average value from the
                                          ; measured value
        CMP     #028h,res_r               ; is the detected signal x cycles long?
        JGE     invalid_bit               ; restart detection, this is not a valid
                                          ; sequence
        JHS     no_start_bit
        INCD    RSTAT                     ; go to RSTATE 2, Data Reception, Start Bit
                                          ; detected
        BIC     #CCIE,&CCTL2              ; disable CCR2 interrupt
        BIC     #022h,&TACTL              ; stop Timer_A and disable interrupt

no_start_bit
        INC     wake_up_counter           ; count the pulses of the trainings sequence,
                                          ; to terninate at least after 8ms
        CMP     #02Fh,wake_up_counter     ; compare the value of the counter with the
                                          ; maximum value of the pulses of the trainings
                                          ; sequence
        JGE     invalid_bit               ;
        RETI

invalid_bit
        CMP     #08h,wake_up_counter      ; to avoid errors during the first run
        JEQ     invalid_bit_end           ; skip clearing RSTAT
        CLR     RSTAT                     ; restart the detection, this is not a valid
                                          ; sequence
        CLR     wake_up_counter           ; initialize the wake_up_counter

invalid_bit_end
        RETI
```

Appendix P. *WDT_INT* — Watchdog Timer Interrupt Routine

WDT_INT generates an exact timing for RF reception and transmission, and for timing the RS232 communication.

```
;******************************* WDT Interrupt Routine ****************************************************
; last change: August 19th 1999
;
; purpose:     generate periodical wakeup for communication routines (timing generation)
;
;**************************************************************************************************************

WDT_INT
        BIC     #CPUOFF,0(SP)                   ; reactivate CPU
        RETI
```

Appendix Q. Interrupt Vector Table

```
;***************************************** Interrupt vector table ********************************************
; last change: 15th June 2000
;
;*************************************************************************************************************

reset
        RSEG    INTVEC                      ;
        DW      START                       ; 0FFE0h not used
        DW      START                       ; 0FFE2h not used
        DW      START                       ; 0FFF4h P1_INT
        DW      START                       ; 0FFE6h I/O Port P2
        DW      START                       ; 0FFE8h not used
        DW      START                       ; 0FFEAh not used
        DW      START                       ; 0FFECh not used
        DW      START                       ; 0FFEEh not used
        DW      TA_INT                      ; 0FFF0h Timer_A, CCIFG1, CCIFG2, TAIFG
        DW      START                       ; 0FFF2h CC0_INT
        DW      WDT_INT                     ; 0FFF4h Watchdog Timer in timer mode
        DW      START                       ; 0FFF6h not used
        DW      START                       ; 0FFF8h not used
        DW      START                       ; 0FFFAh not used
        DW      START                       ; 0FFFCh NMI, Oscillator fault
        DW      START                       ; 0FFFEh Power On Reset, WDTIFG

        END     main
```

Appendix R. Header File

```
/*****************************************************************************************************************
* Special and standard definitions for the MSP-EVKTRF6900 (Revision B),
* used Device:  MSP430F1121
*
* last change:  19.6.2000
*
* Texas Instruments
*
*****************************************************************************************************************/

/*****************************************************************************************************************
* STATUS REGISTER BITS
*****************************************************************************************************************/

#define C           0x0001
#define Z           0x0002
#define N           0x0004
#define V           0x0100
#define GIE         0x0008
#define CPUOFF      0x0010
#define OSCOFF      0x0020
#define SCG0        0x0040
#define SCG1        0x0080

/* Low Power Modes coded with Bits 4-7 in SR */

#ifndef __IAR_SYSTEMS_ICC      /* Begin #defines for assembler              */
#define LPM0        CPUOFF
#define LPM1        SCG0+CPUOFF
#define LPM2        SCG1+CPUOFF
#define LPM3        SCG1+SCG0+CPUOFF
#define LPM4        SCG1+SCG0+OSCOFF+CPUOFF
                               /* End #defines for assembler                */
#else                          /* Begin #defines for C                      */
#define LPM0_bits    CPUOFF
#define LPM1_bits    SCG0+CPUOFF
#define LPM2_bits    SCG1+CPUOFF
#define LPM3_bits    SCG1+SCG0+CPUOFF
#define LPM4_bits    SCG1+SCG0+OSCOFF+CPUOFF

#include "In430.h"

#define LPM0          BIS_SR(LPM0_bits)        /* Enter Low Power Mode 0          */
#define LPM0_EXIT _BIC_SR(LPM0_bits)           /* Exit Low Power Mode 0           */
#define LPM1          _BIS_SR(LPM1_bits)       /* Enter Low Power Mode 1          */
#define LPM1_EXIT  _BIC_SR(LPM1_bits)          /* Exit Low Power Mode 1           */
#define LPM2          _BIS_SR(LPM2_bits)       /* Enter Low Power Mode 2          */
#define LPM2_EXIT  _BIC_SR(LPM2_bits)          /* Exit Low Power Mode 2           */
#define LPM3          _BIS_SR(LPM3_bits)       /* Enter Low Power Mode 3          */
#define LPM3_EXIT  _BIC_SR(LPM3_bits)          /* Exit Low Power Mode 3           */
#define LPM4          _BIS_SR(LPM4_bits)       /* Enter Low Power Mode 4          */
#define LPM4_EXIT  _BIC_SR(LPM4_bits)          /* Exit Low Power Mode 4           */
```

```
#endif                                          /* End #defines for C            */

/************************************************************************************************
* PERIPHERAL FILE MAP
************************************************************************************************/

/************************************************************************************************
* SPECIAL FUNCTION REGISTER ADDRESSES + CONTROL BITS
************************************************************************************************/

#define IE1_             0x0000                 /* Interrupt Enable 1            */
sfrb    IE1             = IE1_;
#define WDTIE            0x01
#define OFIE             0x02
#define NMIIE            0x10
#define ACCVIE           0x20

#define IFG1_            0x0002                 /* Interrupt Flag 1              */
sfrb    IFG1            = IFG1_;
#define WDTIFG           0x01
#define OFIFG            0x02

#define NMIIFG           0x10

#define ME1_             0x0004                 /* Module Enable 1               */
sfrb    ME1             = ME1_;

#define IE2_             0x0001                 /* Interrupt Enable 2            */
sfrb    IE2             = IE2_;

#define IFG2_            0x0003                 /* Interrupt Flag 2              */
sfrb    IFG2            = IFG2_;

#define ME2_             0x0005                 /* Module Enable 2               */
sfrb    ME2             = ME2_;

/************************************************************************************************
* WATCHDOG TIMER
************************************************************************************************/

#define WDTCTL_          0x0120                 /* Watchdog Timer Control        */
sfrw    WDTCTL           = WDTCTL_;
/* The bit names have been prefixed with "WDT" */
#define WDTIS0           0x0001
#define WDTIS1           0x0002
#define WDTSSEL          0x0004
#define WDTCNTCL         0x0008
#define WDTTMSEL         0x0010
#define WDTNMI           0x0020
#define WDTNMIES         0x0040
#define WDTHOLD          0x0080

#define WDTPW            0x5A00
```

```
/* WDT-interval times [1ms] coded with Bits 0-2 */
/* WDT is clocked by fMCLK (assumed 1MHz) */
#define WDT_MDLY_32         WDTPW+WDTTMSEL+WDTCNTCL
                                                /* 32ms interval (default) */
#define WDT_MDLY_8          WDTPW+WDTTMSEL+WDTCNTCL+WDTIS0
                                                /* 8ms     " */
#define WDT_MDLY_0_5        WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1
                                                /* 0.5ms   " */
#define WDT_MDLY_0_064      WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1+WDTIS0
                                                /* 0.064ms " */
/* WDT is clocked by fACLK (assumed 32KHz) */
#define WDT_ADLY_1000       WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL
                                                /* 1000ms  " */
#define WDT_ADLY_250        WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS0
                                                /* 250ms   " */
#define WDT_ADLY_16         WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS1
                                                /* 16ms    " */
#define WDT_ADLY_1_9    WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS1+WDTIS0
                                                /* 1.9ms   " */
/* Watchdog mode -> reset after expired time */
/* WDT is clocked by fMCLK (assumed 1MHz) */
#define WDT_MRST_32         WDTPW+WDTCNTCL
                                                /* 32ms interval (default) */
#define WDT_MRST_8          WDTPW+WDTCNTCL+WDTIS0
                                                /* 8ms     " */
#define WDT_MRST_0_5        WDTPW+WDTCNTCL+WDTIS1
                                                /* 0.5ms   " */
#define WDT_MRST_0_064      WDTPW+WDTCNTCL+WDTIS1+WDTIS0
                                                /* 0.064ms " */
/* WDT is clocked by fACLK (assumed 32KHz) */
#define WDT_ARST_1000       WDTPW+WDTCNTCL+WDTSSEL
                                                /* 1000ms  " */
#define WDT_ARST_250        WDTPW+WDTCNTCL+WDTSSEL+WDTIS0
                                                /* 250ms   " */
#define WDT_ARST_16         WDTPW+WDTCNTCL+WDTSSEL+WDTIS1
                                                /* 16ms    " */
#define WDT_ARST_1_9        WDTPW+WDTCNTCL+WDTSSEL+WDTIS1+WDTIS0
                                                /* 1.9ms   " */

/* INTERRUPT CONTROL */
/* These two bits are defined in the Special Function Registers */
/* #define WDTIE            0x01 */
/* #define WDTIFG           0x01 */

/***********************************************************************************************************
* DIGITAL I/O Port1/2
***********************************************************************************************************/

#define     P1IN_           0x0020          /* Port 1 Input                 */
const sfrb  P1IN            = P1IN_;
#define     P1OUT_          0x0021          /* Port 1 Output                */
sfrb        P1OUT           = P1OUT_;
#define     P1DIR_          0x0022          /* Port 1 Direction             */
sfrb        P1DIR           = P1DIR_;
```

```
#define     P1IFG_          0x0023        /* Port 1 Interrupt Flag          */
sfrb        P1IFG          = P1IFG_;
#define     P1IES_         0x0024         /* Port 1 Interrupt Edge Select   */
sfrb        P1IES          = P1IES_;
#define     P1IE_          0x0025         /* Port 1 Interrupt Enable        */
sfrb        P1IE           = P1IE_;
#define     P1SEL_         0x0026         /* Port 1 Selection               */
sfrb        P1SEL          = P1SEL_;

#define     P2IN_          0x0028         /* Port 2 Input                   */
const sfrb  P2IN           = P2IN_;
#define     P2OUT_         0x0029         /* Port 2 Output                  */
sfrb        P2OUT          = P2OUT_;
#define     P2DIR_         0x002A         /* Port 2 Direction               */
sfrb        P2DIR          = P2DIR_;
#define     P2IFG_         0x002B         /* Port 2 Interrupt Flag          */
sfrb        P2IFG          = P2IFG_;
#define     P2IES_         0x002C         /* Port 2 Interrupt Edge Select   */
sfrb        P2IES          = P2IES_;
#define     P2IE_          0x002D         /* Port 2 Interrupt Enable        */
sfrb        P2IE           = P2IE_;
#define     P2SEL_         0x002E         /* Port 2 Selection               */
sfrb        P2SEL          = P2SEL_;

/*****************************************************************************************************************
* Timer_A
*****************************************************************************************************************/

#define     TAIV_          0x012E         /* Timer A Interrupt Vector Word              */
sfrw        TAIV           = TAIV_;

#define     TACTL_         0x0160         /* Timer A Control                            */
sfrw        TACTL          = TACTL_;
#define     CCTL0_         0x0162         /* Timer A Capture/Compare Control 0          */
sfrw        CCTL0          = CCTL0_;
#define     CCTL1_         0x0164         /* Timer A Capture/Compare Control 1          */
sfrw        CCTL1          = CCTL1_;
#define     CCTL2_         0x0166         /* Timer A Capture/Compare Control 2          */
sfrw        CCTL2          = CCTL2_;
#define     CCTL3_         0x0168         /* Timer A Capture/Compare Control 3          */
sfrw        CCTL3          = CCTL3_;
#define     CCTL4_         0x016A         /* Timer A Capture/Compare Control 4          */
sfrw        CCTL4          = CCTL4_;

#define     TAR_           0x0170         /* Timer A                                    */
sfrw        TAR            = TAR_;
#define     CCR0_          0x0172         /* Timer A Capture/Compare 0                  */
sfrw        CCR0           = CCR0_;
#define     CCR1_          0x0174         /* Timer A Capture/Compare 1                  */
sfrw        CCR1           = CCR1_;
#define     CCR2_          0x0176         /* Timer A Capture/Compare 2                  */
sfrw        CCR2           = CCR2_;
#define     CCR3_          0x0178         /* Timer A Capture/Compare 3                  */
sfrw        CCR3           = CCR3_;
```

```
#define    CCR4_          0x017A        /* Timer A Capture/Compare 4             */
sfrw         CCR4          = CCR4_;

#define      TASSEL2       0x0400        /* to distinguish from UART SSELx        */
#define      TASSEL1       0x0200
#define      TASSEL0       0x0100
#define      ID1           0x0080
#define      ID0           0x0040
#define      MC1           0x0020
#define      MC0           0x0010
#define      TACLR         0x0004
#define      TAIE          0x0002
#define      TAIFG         0x0001

#define      MC_0          00*10h
#define      MC_1          01*10h
#define      MC_2          02*10h
#define      MC_3          03*10h
#define      ID_0          00*40h
#define      ID_1          01*40h
#define      ID_2          02*40h
#define      ID_3          03*40h
#define      TASSEL_0      00*100h
#define      TASSEL_1      01*100h
#define      TASSEL_2      02*100h
#define      TASSEL_3      03*100h
#define      CM1           0x8000
#define      CM0           0x4000
#define      CCIS1         0x2000
#define      CCIS0         0x1000
#define      SCS           0x0800
#define      SCCI          0x0400
#define      CAP           0x0100
#define      OUTMOD2       0x0080
#define      OUTMOD1       0x0040
#define      OUTMOD0       0x0020
#define      CCIE          0x0010
#define      CCI           0x0008
#define      OUT           0x0004
#define      COV           0x0002
#define      CCIFG         0x0001

#define      OUTMOD_0      00*20h
#define      OUTMOD_1      01*20h
#define      OUTMOD_2      02*20h
#define      OUTMOD_3      03*20h
#define      OUTMOD_4      04*20h
#define      OUTMOD_5      05*20h
#define      OUTMOD_6      06*20h
#define      OUTMOD_7      07*20h
#define      CCIS_0        00*1000h
#define      CCIS_1        01*1000h
#define      CCIS_2        02*1000h
#define      CCIS_3        03*1000h
```

```
#define     CM_0        00*4000h
#define     CM_1        01*4000h
#define     CM_2        02*4000h
#define     CM_3        03*4000h

#define     CLEAR       0x0004      /* clear the Timer_A                 */
#define     CMPOS       0X4000      /* select rising edge                */
#define     CMNEG       0x8000      /* select falling edge               */
#define     CMANY       0xC000      /* select every edge                 */
#define     CONTUP      0x0020      /* continuous up mode                */
#define     MCLK        0x0200      /* select MCLK as clokc source       */

/****************************************************************************************************
* Basic Clock Module
****************************************************************************************************/

#define     DCOCTL_     0x0056      /* DCO Clock Frequency Control       */
sfrb        DCOCTL      = DCOCTL_;
#define     BCSCTL1_    0x0057      /* Basic Clock System Control 1      */
sfrb        BCSCTL1     = BCSCTL1_;
#define     BCSCTL2_    0x0058      /* Basic Clock System Control 2      */
sfrb        BCSCTL2     = BCSCTL2_;
#define     MOD0        0x01
#define     MOD1        0x02
#define     MOD2        0x04
#define     MOD3        0x08
#define     MOD4        0x10
#define     DCO0        0x20
#define     DCO1        0x40
#define     DCO2        0x80

#define     RSEL0       0x01
#define     RSEL1       0x02
#define     RSEL2       0x04
#define     XT5V        0x08
#define     DIVA0       0x10
#define     DIVA1       0x20
#define     XTS         0x40
#define     XTOFF       0x80

#define     DCOR        0x01
#define     DIVS0       0x02
#define     DIVS1       0x04
#define     SELS        0x08
#define     DIVM0       0x10
#define     DIVM1       0x20
#define     SELM0       0x40
#define     SELM1       0x80

/****************************************************************************************************
* Flash Memory
****************************************************************************************************/

#define     FCTL1_      0x0128    /* FLASH Control 1               */
```

```
sfrw        FCTL1          = FCTL1_;
#define     FCTL2_        0x012A         /* FLASH Control 2                    */
sfrw        FCTL2         = FCTL2_;
#define     FCTL3_        0x012C         /* FLASH Control 3                    */
sfrw        FCTL3         = FCTL3_;

#define     FRKEY         0x9600
#define     FWKEY         0xA500
#define     FXKEY         0x3300         /* for use with XOR instruction        */

#define     ERASE         0x0002
#define     MERAS         0x0004
#define     WRT           0x0040
#define     SEGWRT        0x0080

#define     FN0           0x0001
#define     FN1           0x0002
#define     FN2           0x0004
#define     FN3           0x0008
#define     FN4           0x0010
#define     FN5           0x0020
#define     FSSEL0        0x0040         /* to distinguish from UART SSELx    */
#define     FSSEL1        0x0080

#define     BUSY          0x0001
#define     KEYV          0x0002
#define     ACCVIFG       0x0004
#define     WAIT          0x0008
#define     LOCK          0x0010
#define     EMEX          0x0020

/***********************************************************************************************************
* Comparator A
***********************************************************************************************************/

#define     CACTL1_       0x0059        /* Comparator A Control 1              */
sfrb        CACTL1        = CACTL1_;
#define     CACTL2_       0x005A  /* Comparator A Control 2                  */
sfrb        CACTL2        = CACTL2_;
#define     CAPD_         0x005B  /* Comparator A Port Disable               */
sfrb        CAPD          = CAPD_;
#define     CAIFG         0x01
#define     CAIE          0x02
#define     CAIES         0x04
#define     CAON          0x08
#define     CAREF0        0x10
#define     CAREF1        0x20
#define     CARSEL        0x40
#define     CAEX          0x80

#define     CAOUT         0x01
#define     CAF           0x02
#define     P2CA0         0x04
#define     P2CA1         0x08
```

```
#define     CACTL24      0x10
#define     CACTL25      0x20
#define     CACTL26      0x40
#define     CACTL27      0x80

#define     CAPD0        0x01
#define     CAPD1        0x02
#define     CAPD2        0x04
#define     CAPD3        0x08
#define     CAPD4        0x10
#define     CAPD5        0x20
#define     CAPD6        0x40
#define     CAPD7        0x80

/**********************************************************************************************************
* Interrupt Vectors (offset from 0xFFE0)
**********************************************************************************************************/

#define     WDT_VECTOR          10 * 2      /* 0xFFF4 Watchdog Timer                   */
#define     NMI_VECTOR          14 * 2      /* 0xFFFC Non-maskable                     */
#define     RESET_VECTOR        15 * 2      /* 0xFFFE Reset [Highest Priority]         */

#define     PORT1_VECTOR        2 * 2       /* 0xFFE4 Port 1                           */
#define     PORT2_VECTOR        3 * 2       /* 0xFFE6 Port 2                           */
#define     TIMERA1_VECTOR      8 * 2       /* 0xFFF0 Timer A CC1-2, TA                */
#define     TIMERA0_VECTOR      9 * 2       /* 0xFFF2 Timer A CC0                      */
#define     COMPARATORA_VECTOR  11 * 2         /* 0xFFF6 Comparator A              */

/**********************************************************************************************************
* Port 1 definition for the MSP-EVKTRF6900
**********************************************************************************************************/

#define     data         0x0080   /* P1.7 DATA, programming data into the TRF6900    */
#define     SDA          0x0080   /* P1.7 SDA, SDA line for IIC-LCD                   */
#define     clk          0x0040   /* P1.6 CLK, programming clock for the TRF6900      */
#define     SCL          0x0040   /* P1.6 SCL, SCL line for the IIC-LCD               */
#define     strobe       0x0020   /* P1.5 STROBE, STROBE line to the TRF6900          */
#define     tx           0x0010   /* P1.4 TXDATA, transmit data to the TRF6900        */
#define     rx           0x0008   /* P1.3 RXDATA, receive data from TRF6900           */
#define     rxd          0x0004   /* P1.2 RXD_MSP, receive data from RS232            */
#define     txd          0x0002   /* P1.1 TXD_MSP, sent data to the PC via RS232      */
#define     stdb_rs232   0x0001   /* P1.0 RS232 standby mode (when set)               */

/**********************************************************************************************************
* Port 2 definition for the MSP-EVKTRF6900
**********************************************************************************************************/

#define     stdb_trf6900 0x0020   /* P2.5 Standby Mode for the TRF6900                */
#define     lockdet      0x0010   /* P2.4 Lock Detect, the PLL has locked at the
                                     selected frequency                               */
#define     LED3         0x0008   /* P2.3 System Mode LED                             */
#define     rts          0x0004   /* P2.2 request to send (RS232)                     */
#define     mode         0x0002   /* P2.1 Mode 1 or Mode 0                            */
#define     cts          0x0001   /* P2.0 clear to send (RS232)                       */
```

```
/****************************************************************************************************************
* SYSTEM DEFINITIONS
****************************************************************************************************************/

#define     MSTOP       0x0080   /* Stop Mode                                            */
#define     MCONT       0x0020   /* Continuous Mode                                      */

/****************************************************************************************************************
* Basic Clock Module
****************************************************************************************************************/

#define     DIVA_2      0x0010   /* divide ACLK by 2                                     */

/****************************************************************************************************************
* Register Definitions
****************************************************************************************************************/

#define     data_r          R4   /* current transmitted or received byte                 */
#define     word_trf        R4   /* programming buffer for the TRF6900
                                 registers                                               */
#define     bits_r          R5   /* bit counter, counts the bits of the current
                                 byte                                                    */
#define     address_r       R6   /* pointer to RAM address of the current
                                 received or transmitted byte                            */
#define     word_h          R6   /* the high byte of the TRF6900 programming
                                 word                                                    */
#define     word_l          R7   /* the low byte of the TRF6900 programming
                                 word                                                    */
#define     address_start   R7   /* start addres for transmission                        */
#define     RSTAT           R8   /* state of the reception                               */
#define     chcksum_r       R8   /* checksum of the received data                        */
#define     chcksum_s       R9   /* checksum of the transmitted data                     */
#define     wait_r          R9   /* counter register for all waiting loops
                                 (no collusion with res_new_r possible)                  */
#define     wake_up_counter R10      /* counter for valid pulses during training
                                 phase                                                   */
#define     counter         R10  /* universal counter                                    */
#define     current         R11  /* buffer for the current result of Timer_A             */
#define     previous        R12  /* buffer for the previous result of Timer_A            */
#define     res_old_r       R13  /* old value of the CCR2                                */
#define     res_new_r       R14  /* new value of the CCR2
                                 difference gives pulse width -> res_r (both
                                 only during the RF reception)                           */
#define     res_r           R15  /* width of the recently received puls, only
                                 during RF reception                                     */
#define     tr_counter      R15  /* counter for the training sequence                    */
#define     rs232_counterR15     /* used for the detection of the high and low
                                 byte during the rs232 reception                         */
```

```
/*****************************************************************************************************************
* Bits
*****************************************************************************************************************/
/****************************************************************************************************************/

#define        data              0x0080

/*****************************************************************************************************************
* Clock
*****************************************************************************************************************/

#define        clock_new_1   0x0070        /* ACLK 1/8 from external crystal                      */
#define        clock_new_2   0x00C8        /* MCLK directly from LFXTCL
                                           SMCLK directly from MCLK                               */

/*****************************************************************************************************************
* Watchdog Timer
*****************************************************************************************************************/

#define        WDT26MS       0x0005        /* set the timing to 26 microseconds                   */

/*****************************************************************************************************************
* Other Registers for RF transmission
*****************************************************************************************************************/

sfrw           data_rx_state   = 0x0248;   /* stores the state of the RF and RS232                */
                                           /* reception                                           */
```

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 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.

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.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265