



Application Note 54

MICRF505/506 Basic: Handling the Data Interface

Author: Per Kristian Bergseth

Who Should Read this Application Note?

This application note is intended for users of model MICRF505/MICRF506 radio transceivers, with emphasis on firmware design.

Summary

This application note emphasizes the microcontroller point-of-view when working with the MICRF505/MICRF506 radio transceiver. Using the "data interface" and transmitting/receiving bits is described.

A troubleshooting section is included.

Key Words

Data Interface, DC Component, Bits and Elements, On-chip Clock Recovery, DC-free Coding, Manchester Code, 3B4B Code

Introduction

In the following, "MICRF transceiver" is used as an abbreviation for the "model MICRF505/MICRF506 radio transceiver." "MCU" is an abbreviation for "micro controller unit".

The MICRF transceiver is typically controlled by an MCU. The most basic task is to program the MICRF transceiver, that is, setting the MICRF transceiver in transmit or receive mode, the RF frequency, bit rate and so on. The programming is done by way of the "control interface," and this task is described in a separate application note (AN-46). A detailed description of the fields to program into the MICRF transceiver is given in the appropriate MICRF transceiver data sheet.

The focus of this application note is the ability to transmit and receive data at the bit- or byte-level.

After transmitting/receiving bits and bytes, the next task is to pack or unpack the bits and bytes into/from a "frame", and making a "packet engine".

Key elements in this application note:

- Data interface to the MICRF transceiver
- Using the built-in clock recovery feature ("Sync_en" field)
- Data coded with or without a DC-component

Data Interface to MICRF

The "data interface" is used to transmit and receive data bits. This is a separate interface from the MICRF505/MICRF506 ("MICRF"). It is independent of the control interface used to program the MICRF transceiver (refer to application note AN-46).

Two pins are used in the "data interface": "DATAIXO" and "DATACLK". Note that the DATACLK pin is optional and is used in "bit synchronized mode" (bit-sync mode) only, that is, when the "Sync_en" field is "1". In bit-sync mode, the MICRF uses the on-chip clock recovery feature while in receive (rx) mode and provides a data clock (DATACLK) in both receive (rx) and transmit (tx) modes (refer to "Built-in Bit Synchronizer").

The DATACLK pin is always an output from the MICRF. That is, the MCU is a "slave" in the data interface. Note that the MCU is a "master" in the separate control interface.

The bit rate (the frequency of the DATACLK signal) is determined by several fields in the control word. The BitRate_clkS, BitSync_clkS and RefClk_K fields must be set (refer to "Control Word Settings for Bit Rate").

The Input/Output pins of the data interface are summarized in Table 1.

Pin Name	Function	Direction MCU MICRF	Comment
DATAIXO	Data In/Out	← →	Output from MCU only while transmitting, else: Input to MCU
DATACLK	Data clock	←	Always input to MCU

Table 1. The 2-Wire Data Interface.

The MCU pin connected to DATAIXO must be able to switch between input and output states. Alternatively, it is possible to use one MCU pin for data in and one MCU pin for data out, and to enable or disable these pins with external components and control-signals from the MCU. This requires more pins and added hardware (in any case, there is only one MICRF data In/Out pin).

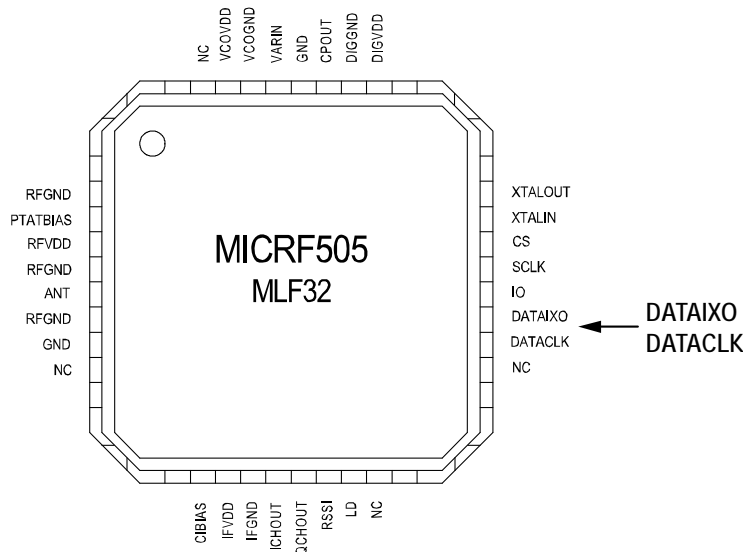


Figure 1. MICRF505/506 Transceiver Pin Assignment.

Using the data interface:

- When transmitting bits:
 - Ensure that DATAIXO is an output from the MCU
 - Repeat for all bits:
 - At the negative edge of DATACLK, bring DATAIXO high to transmit a “1”, bring DATAIXO low to transmit a “0”
 - Make sure DATAIXO is an input when all bits are completely transmitted
- When receiving bits:
 - Make sure DATAIXO is an input to the MCU
 - Repeat for all bits:
 - At the negative edge of DATACLK, read DATAIXO state; read “1” if it is high, “0” if low.

This is described further in the text below.

Transmit-mode:

- Transmit-mode is selected by setting the “Mode” field in the control word to “Transmit” (binary “11”).
- In transmit-mode, the frequency dividers in “A0,N0,M0” are used. If the modulation type is “Switch between two sets of A,N,M dividers”, then the “A1,N1,M1” set is used as well. Refer to “Modulation Type”.

- The MICRF transceiver samples the DATAIXO line at the positive edges of DATACLK. Then, if an SPI-like hw-module is used, make sure that the MCU updates DATAIXO at negative edges of DATACLK. If a bit-by-bit (“bit-banging”) approach is used, then the DATAIXO line can be changed at any time, as long as DATAIXO is stable when a positive edge is made on DATACLK. The MICRF provides the DATACLK signal.
- If “bit-banging” at high data rates (for example, 200kbps), the number of MCU instruction cycles per bit may be an issue. For maximum efficiency, change the DATAIXO line as soon as possible after a positive edge (at the moment when the DATAIXO line is sampled by the MICRF transceiver, the “time to next sampling” is maximum and close to one bit length).
- In general, make sure the DATAIXO line is an output during transmission – and only then. This is of particular importance if the selected modulation type requires a DC-free signal. More details are given in the next sections.
- Make sure that the last bit is completely sent before leaving transmit-mode.
- Figure 2 illustrates how bits are transmitted. In the graphic, the DATAIXO line is changed (by the MCU) at negative edges of the DATACLK-pulses (made by the MICRF).

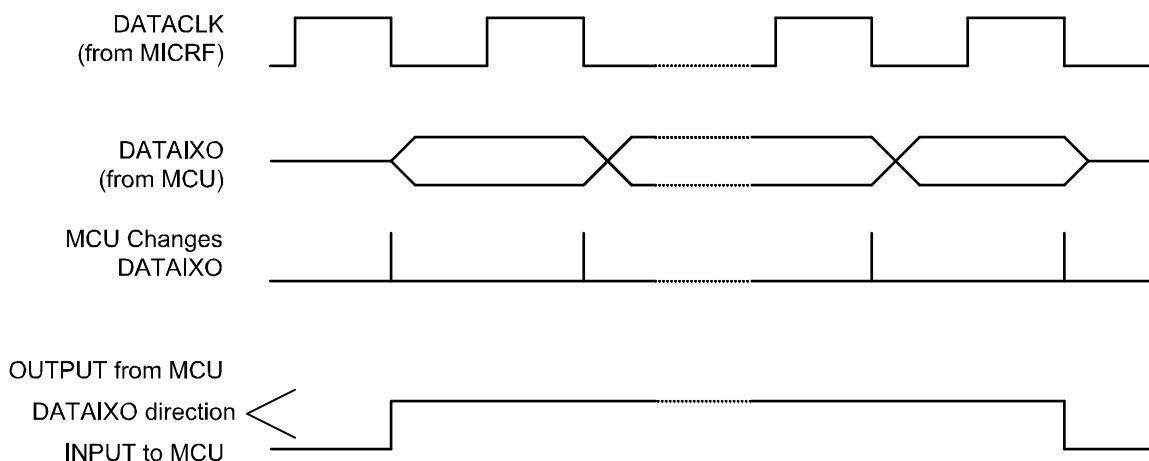


Figure 2. Data Interface in Transmit Mode

The DATAIXO line is bi-directional; it is an output from the MICRF transceiver in rx-mode and an input to the MICRF transceiver in tx-mode.

Note: To avoid a situation where both the MICRF and MCU are driving the line (i.e. an output-to-output case), make the DATAIXO an output from the MCU when starting to transmit the first bit, bring it back into tri-state (input) when the last bit is completely transmitted.

In addition to avoid the output-to-output case, this rule is needed for another situation:

If closed-loop VCO modulation is used, then the DATAIXO line must be kept in tri-state until modulation starts or else the signal will have a DC-component.

Receive Mode:

- Receive-mode (see Figure 3) is selected by setting the “Mode” field in the control word to “Receive” (binary “10”).
- In receive-mode, the frequency dividers in “A0,N0,M0” is used. An exception to this rule is described in “Modulation Type”.
- The MICRF transceiver changes the DATAIXO line at the positive edges of DATACLK. Then, if a SPI-like hw-module is used, make sure that the MCU reads DATAIXO at negative edges of DATACLK. If a bit-by-bit (“bit-banging”) approach is used, then the DATAIXO line can be read at any time, as long as DATAIXO is stable when you read it (that is, it has reached a valid “1” or “0”).
- If “bit-banging” at high data rates (for example, 200kbps), the number of MCU instruction cycles / bit may be an issue. For maximum efficiency, start the read sequence (for example, enter an interrupt service routine) as soon as possible after a positive edge. If this method is used, make sure that the DATAIXO is stable “1” or “0” when you actually sample it.
- Figure 3 shows how bits are received. In this figure, the DATAIXO line is sampled (by the MCU) at negative edges of the DATACLK-pulses provided by the MICRF transceiver.

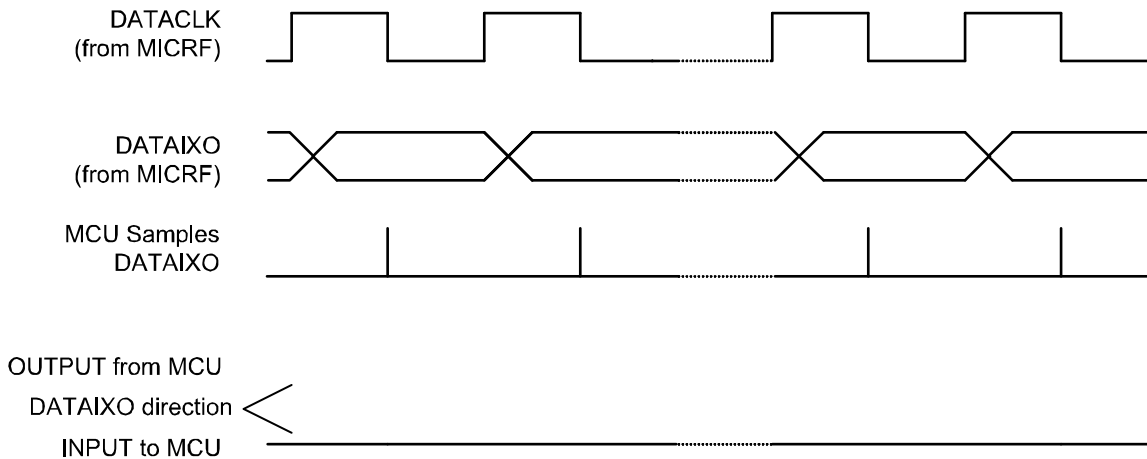


Figure 3. Data Interface in Receive Mode.

Finally, observe the following property of the data interface in rx-mode:

In rx-mode, the MICRF transceiver will continuously demodulate the RF input and give out “bits” to the user, even if there is no transmitter present. The user must detect “start-of-frame,” that is, separate noise from data signals. This is a task for the “Packet engine” described in Application Note 57.

Control Word Settings for Bit Rate

If the “Sync_en” field in the control word is “1”, then the MICRF transceiver will generate a “data clock” (DATACLK pin) in both receive and transmit mode. If the “Sync_en” field is “0”, then no clock is generated on this pin.

Internally to the RF chip, two “clocks” are made: The “BitRate clock” and the “BitSync clock”.

In transmit mode, the frequency (in Hz) of the “BitRate clock” (“ $f_{\text{BITRATE_CLK}}$ ”) should be set equal to the number of bits to be tx’ed on-the-air. Example: For 20kbps, the “ $f_{\text{BITRATE_CLK}}$ ” should be 20kHz. The other clock, “BitSync clock”, is not used in transmit mode.

In receive mode, the frequency (in Hz) of the “BitSync clock” (“ $f_{\text{BITSYNC_CLK}}$ ”) should be set equal to (16 * the number of on-the-air bits/second to be rx’ed). Example: For 20kbps, the “ $f_{\text{BITSYNC_CLK}}$ ” should be $16 \times 20 = 320$, i.e. 320kHz. The other clock, “BitRate clock” is not used in receive mode.

The formulas for the clocks are given below. The PC-program “RF TestBench” can be used to calculate all the fields in the control word.

$$f_{\text{BITRATE_CLK}} = \frac{f_{\text{XCO}}}{\text{RefClk_K} \times 2^{(7-\text{BitRate_clkS})}}$$

$$f_{\text{BITSYNC_CLK}} = \frac{f_{\text{XCO}}}{\text{RefClk_K} \times 2^{(7-\text{BitSync_clkS})}}$$

f_{XCO} is the crystal oscillator frequency (the crystal connected to the RF chip)

RefClk_K, BitRate_clkS and BitSync_clkS are fields in the control word:

- Value of RefClk_K: 1 – 63
- Value of BitRate_clkS: 0 – 6
- Value of BitSync_clkS: 0 – 6 (MICRF506: 0 – 7, do not use “7” for MICRF505)

In transmit mode, the number of transmitted bits per second is equal to the $f_{\text{BITRATE_CLK}}$ frequency (in Hz). In receive mode, $f_{\text{BITSYNC_CLK}}$ should be 16 times higher than the bit rate.

In transmit mode, if the built-in modulator is used (i.e. VCO modulation), then the RefClk_K field is used in $f_{\text{MOD_CLK}}$ as well:

$$f_{\text{MOD_CLK}} = \frac{f_{\text{XCO}}}{\text{RefClk_K} \times 2^{(7-\text{Mod_clkS})}}$$

$f_{\text{MOD_CLK}}$ should be eight times (or more) higher than $f_{\text{BITRATE_CLK}}$.

- Value of Mod_clkS: 0 – 7

The PC program “RF TestBench” can be used to calculate all the fields in the control word.

RF TestBench first calculates RefClk_K and BitRate_clkS, and then it finds BitSync_clkS and Mod_clkS – keeping RefClk_K unchanged:

$$\begin{aligned} \text{BitSync_clkS} &= \text{BitRate_clkS} + 4, \\ \text{Mod_clkS} &= \text{BitRate_clkS} + 3, \end{aligned}$$

If BitSync_clkS and Mod_clkS are calculated this way, then:

$$f_{\text{BITSYNC_CLK}} = 16 \times f_{\text{BITRATE_CLK}}$$

and:

$$f_{\text{MOD_CLK}} = 8 \times f_{\text{BITRATE_CLK}}$$

In this way, it will not be necessary to change these fields when switching between rx and tx mode – but the maximum value of the BitRate_clkS field is limited to 2 (MICRF505) or 3 (MICRF506) (then BitSync_clkS will be 6 or 7, which is the maximum value for this field). When switching between receive and transmit, it is in most cases desirable to only change the «Mode» field (and A,N,M fields to change frequency).

The user should note that a better resolution might be achievable if the RefClk_K field is changed when switching between transmit and receive. Then, with VCO modulation, BitRate_clkS can have a maximum value of 4 (making Mod_clkS = 7). With A, N, M modulation, BitRate_clkS can be maximum 6. Again, note that the price of this increased flexibility is that more fields than “Mode” (and A,N,M dividers) must be programmed into the MICRF transceiver when changing mode of operation. In most applications, the fields described here can be kept unchanged when switching between receive and transmit mode.

If a specific bit rate can't be reached, the user might consider to change the crystal. Note that this will affect RF frequencies as well as bit rates.

Some common misunderstandings related to the topics described in this section:

- The user should note that the “crystal frequency” in this section refers to the crystal connected to the RF chip. In many applications, a separate crystal is connected to the MCU. Make sure to use the correct crystal frequency (of the “RF crystal”) when calculating RF parameters and bit rates.
- When discussing “clocks”, the user should also note that the RF chip starts in power-down mode after power is applied, with the crystal oscillator off. The MICRF transceiver must be programmed to standby or to receive/transmit mode before the crystal oscillator signal becomes active. If a common crystal for both the MICRF transceiver and MCU is going to be used, this should be taken into consideration.

Built-in Bit Synchronizer

MICRF505/MICRF506 transceivers have a built-in bit synchronizer feature which is enabled by setting the "Sync_en" field = "1". Using the bit synchronizer is recommended for the following reasons:

- In receive mode, the built-in clock recovery mechanism will oversample the demodulated data signal (16 samples/bit) and clock out the resulting "0" or "1". The effect is that no glitches will appear in the bit given on the DATAIXO line; the DATAIXO line will be constant high or low for the duration of any given bit.
- If data bits are clocked in and out of the MICRF transceiver, then a high-ppm crystal or oscillator may be used for the MCU; the MCU does not have to construct bits with an accurate length or sample incoming bits at an accurate rate. Note that the MCU may still need to use a low-ppm crystal to accomplish other tasks.
- If the MCU has a built-in hw module like "edge detection" or "serial port", then the DATACLK pin can be used to trigger an interrupt for "edge detected - send/sample data", or it can be the clock source for a serial module, like "SPI."
- The sensitivity figures in the data sheet are based on Sync_en = 1. If the built-in bit synchronizer is not used, the sensitivity is lower (as much as 3-6 dB).
- If the "transmitted bit rate" is in the "programmed received bit rate" $\pm 2,5\%$ range, the synchronizer will still work.
- Note that the bit-synchronizer needs to adjust itself to the in-coming RF signal. The transmitting RF unit should start it's transmission with a "preamble" (= "learning sequence") of typically 24 (minimum 21) 1010... bits.

How to use it:

- Set Sync_en = 1
- Set the BitRate clock = the wanted on-the-air bitrate
- Set the BitSync clock = 16* BitRate_Clk
- The PC-tool "RF TestBench" can be used to calculate all fields
- Observe that DATACLK is always an output from the MICRF transceiver, i.e. an input to the MCU

Some applications may select to not use the bit synchronizer:

- If the application MCU is short of IOs (not enough pins to include the DATACLK)
- If the bit rate can't be reached accurately enough by the MICRF transceiver bit rate generator

- If the data to transmit is provided directly from an external source, not from the MCU programming the MICRF transceiver
- If "tx and rx like an UART", i.e., "start-stop formatted bytes" or "asynchronous data"

Using Sync_en = 1, but not using DATACLK:

- If the application is short of IOs, and the designer selects not to use DATACLK: It might still be advantageous to use Sync_en = 1 in rx-mode (not in tx-mode, since you in that case do not know when to change the DATAIXO line). In rx-mode, the signal will be over-sampled and a "clean" signal without short noise spikes (that can occur if the signal is weak) will be given out on the DATAIXO line. This will make it easier to generate a "clock recovery procedure" in the MCU (i.e., making a decision for when the DATAIXO line should be sampled in this case).

Modulation Type

This subsection is included in this app note to give the user a background for the necessary line coding methods.

There are two types of modulation:

Using the built-in modulator in "VCO modulation"

Switch between two sets of A,N,M in "A,N,M modulation"

VCO modulation:

- The MICRF transceiver is using A0,N0,M0 in both receive and transmit mode
- Note that a single set of A,N,M dividers is sufficient with this type of modulation.
- The minimum on-the-air bit rate is in the 40kbps range.
- With VCO modulation, a DC free code must be used.
- It is important to keep the DATAIXO line in tristate (input to MCU) until start-of-modulation. Then make DATAIXO an output.
- When all bits are sent, make DATAIXO an input to the MCU again.

A,N,M modulation:

- The MICRF transceiver is using A0,N0,M0 for the "0" frequency, and "A1,N1,M1" for the "1" frequency.
- The MICRF transceiver chip is using "A0,N0,M0" in receive mode (see exception below)
- When going from transmit to receive mode and Sync_en = 1: In rx-mode, the A0, N0, M0 set is used if the last data bit clocked out was "0" or

“tristate” (that is: the state of DATAIXO at the very last clock-pulse before leaving tx-mode). However, if the last data bit clocked out is a “1”, then the A1,N1,M1 set is used. To avoid this, make sure that the DATAIXO is put into tristate when the last data bit is sent, and make sure rx-mode is not entered until 1 bit-time has passed (i.e., CS-pin going low after clocked in rx-word). Alternatively, always program both the “0” and “1” sets with the rx-frequency.

- Note that three sets of A,N,M dividers is needed in this case: For “TX0”, “TX1” and “RX”. The frequency dividers must be re-programmed when going from receive to transmit and vice versa. At a single time, the control-word only holds the A,N,M values for transmit mode or receive mode.
- The maximum on-the-air rate is in the 20kbps range
- With A,N,M modulation, it is not necessary to use a DC-free code.

Data Coded with or without a DC-Component

The terms “bits” and “elements” are used in this subsection. Generally, “bits” are the information a user wants to transmit.

To make the transmitted signal DC-free, it might be necessary to code one bit into two or more “elements”. For example, with Manchester code, every “bit” is coded into two “elements”.

The “on-the-air data rate” is synonymous with “elements txed pr second”. It is this rate that is programmed into model MICRF505 and MICRF506 transceivers.

NRZ_L Code

“NRZ L” means “Non-return-to-zero, level”.

This can be used if a DC-component can be accepted, such as if A,N,M modulation is used. See Figure 4.

This is a 1:1 code, that is:

- If bit to transmit is “1”, then bring DATAIXO line high
- If bit to transmit is “0”, then bring DATAIXO line low

Observe that this code will, in general, have a DC-component. (Example: Transmit only 1’s).

There is no “code overhead” for this code, meaning that the number of bits-per-second equals the number of elements-per-second.

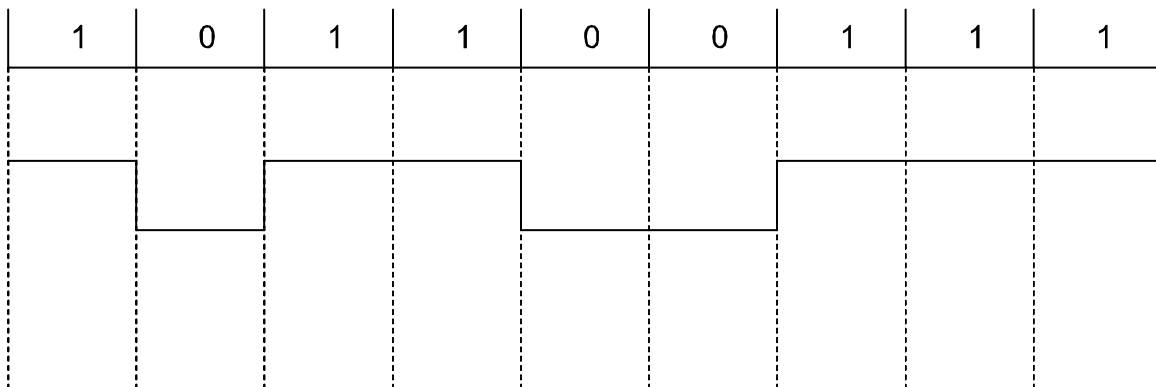


Figure 4. NRZ L Code.

Manchester Code

The Manchester Code, Figure 5, can be used if the transmitted signal must be DC-free, i.e., if VCO modulation is used.

Manchester code is the simplest method to get a DC-free signal. A “1”-bit is coded into the two signal elements “01” and a “0”-bit is coded into the two signal elements “10” (or inversely – both variants exists, and work equally good as long as the transmit (tx) and receive (rx) side use the same code/decode).

Bit	Signal Elements
0	10
1	01

Transmitting a Manchester-Coded signal is straightforward and can be done on-the-fly (i.e. “coding” while physically transmitting):

Repeat for every bit:

- Wait for time to change DATAIXO (can be an interrupt)
- Set DATAIXO high or low (bit is “0” or “1”, resp.)
- Wait for time to change DATAIXO (can be an interrupt)
- Invert DATAIXO

End-repeat

If signal elements can be sent in octets via an MCU built-in serial module (for example, loading a register with eight signal elements), then a simple look-up table can be used. Get the Ms nibble to transmit, Manchester Code it into eight elements and transmit it; then repeat for the Ls nibble of the byte.

Example:

Initially, Manchester_Code[0]...[15] are filled:

Manchester_Code[0] = ‘10101010’

Manchester_Code[1] = ‘10101001’

...

Manchester_Code[15] = ‘01010101’

Repeat for every byte

- Wait for time to update Serial_Data register (can be an interrupt) (Serial_Data is a micro controller register)

- Set Serial_Data = Manchester_Code [MS nibble of the byte]
- Wait for time to update Serial_Data register (can be an interrupt) (Serial_Data is a micro controller register)
- Set Serial_Data = Manchester_Code[LS nibble of the byte]

End_Repeat

Receiving a Manchester-coded signal requires a mechanism to identify the first and second element of a bit.

Typically, the Start-of-Frame field (“FrameSync”) can be used to determine the bit-boundaries (i.e., “this is the first and this is the second element in a bit”). The receiver samples every signal element, and when the FrameSync is found, the next element to read is the first element of a Manchester-coded bit. This is described in the “Low Level RF protocol” application note AN-57.

If “bit-banging” is used and after FrameSync is found, use the following procedure.

Repeat for every bit to read:

- Wait for a new signal element
- Ignore the signal element
- Wait for a new signal element
- Sample and store value of DATAIXO in a register

End-Repeat

If a serial port is used, the serial port-function can be enabled immediately after FrameSync is read. Then Ms and Ls parts of every byte must be decoded.

If it is not applicable to sample every signal element, then a more sophisticated method must be used: Store received signal elements in a buffer, and test if it contains a FrameSync somewhere — the received elements will typically not be byte-aligned, that is, the MCU must find the byte boundaries in the buffer. Then, decode bits from that point. Note that this requires more of the MCU than simply “sampling bits”. However, with this method, the MCU can work on the received elements in the background while filling up “bytes” in an interrupt routine.

The “code overhead” for this code is 100%. That is, the number of elements is two times the number of bits.

Note that the maximum number of equal elements in a row is two.

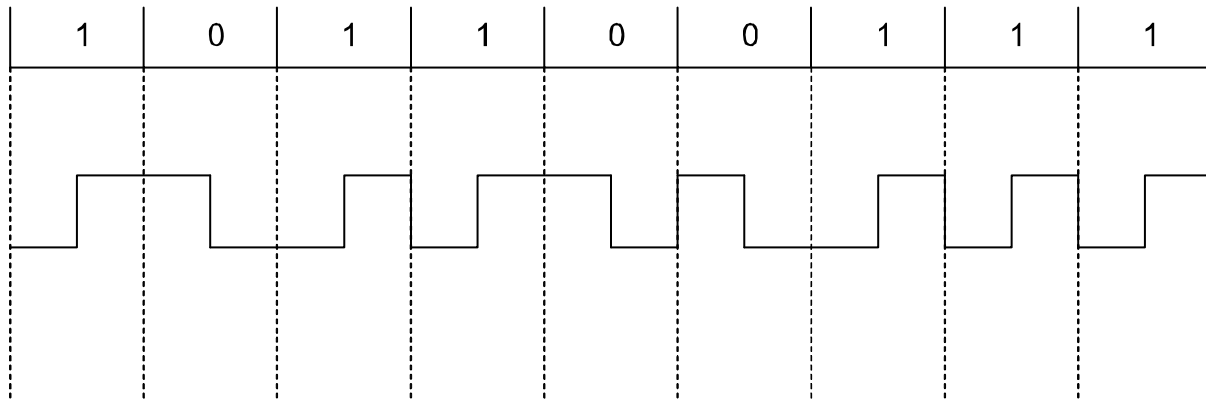


Figure 5. Manchester Code.

3B4B code

3B4B code, see Figure 6, is a *block-code*. Three bits are coded into four signal elements (a more appropriate name with these terms is 3B4E). Manchester Code can be considered a 1B2B (1B2E) code.

The elements are selected to give a DC-free code (on the average). That is, two of the four elements should be “1”, the other two should be “0”. To code three bits, eight combinations must be available (2³). There are only six combinations with two 0’s and two 1’s. The remaining combinations can be one 0 and three 1’s or vice versa, and then alternate between the “three 0’s” and the “three 1’s” bytes, as described in the example below.

Example:

Bits	Signal Elements	Comments
000	1011/0100	Use opposite number of 1’s next time a 000 or 111 is txed
001	1100	
010	0011	
011	1010	
100	0101	
101	1001	
110	0110	
111	1101/0010	Use opposite number of 1’s next time a 000 or 111 is txed

This code requires some pre-coding (which NRZ and Manchester do not, strictly speaking). Before starting to transmit, three bits must be collected and the corresponding four elements constructed. And when these four elements are sent, the next three bits must be available etc.

The “code overhead” for this code is 33 percent. That is: The number of elements is 4/3 times the number of bits. In addition, if (the number of bits)/3 is not an integer, then 1 or 2 dummy-bits must be added.

Max number of equal elements in-a-row is four.

However, three following combinations can be 1100 0100 0011. Worst case is 7:1 in an eight-element sequence.

The code can be used in systems where a DC-free code is essential. But, due to the relatively low code overhead, this code can be used in systems that do not require a DC-free signal, but still require or benefit from rapid transitions (for clock recovery purposes).

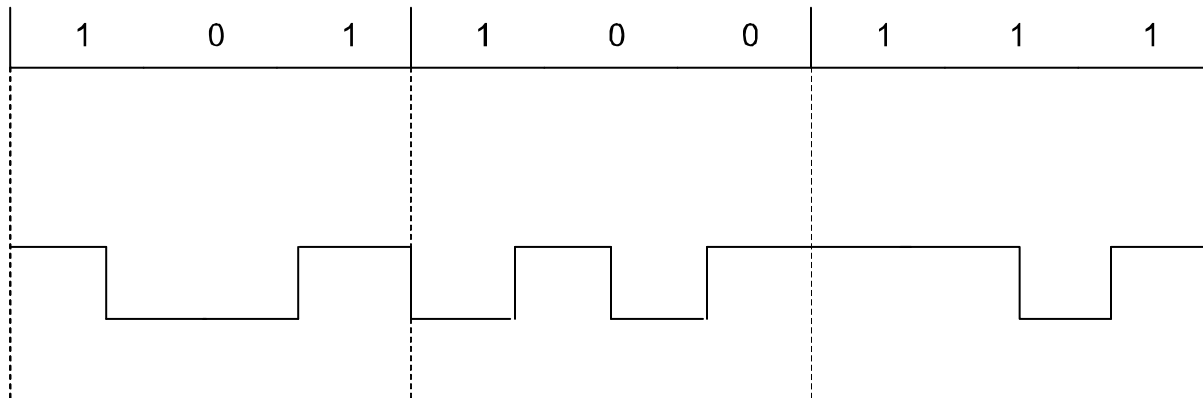


Figure 6. 3B4B Code.

Transmitting like a UART

- A UART (Figure 7) transmits asynchronously, with a defined format - like 9600bps, one start, eight data, no parity and one stop bit (9600-8-N-1)
- A start-bit occurs randomly (the time between bytes is random)
- If this type of signal is applied directly to the MICRF505 transceiver DATAIXO line with Sync_en = 0:
- A,N,M mod: Okay.
- VCO mod: NOT okay, as a DC-free signal is required
- Although start ("0") and stop ("1") bits are included, this is not enough to make it DC free
- To make it DC-free: Specify no delay between bytes, and the eight data bits must be a DC-free pattern
- Transmitting: You cannot use the bit synchronizer (you must make sure that Sync_en = 0). The reason for this is that the input from you will not be synchronized to the clock signal (the UART input is by nature an "asynchronous" signal, not "clocked"). Receiving: Sync_en can be used – in which case you must also include a "training sequence": The clock recovery feature requires 21 (suggested: 24) transitions to be completely synchronized to the incoming signal. If you are transmitting a "very long idle state" (time between two bytes), then the bit-synchronizer in the receiver will probably be offset (it is not adjusted because there are no incoming transitions). And in this case, it will need a number of transitions to re-synchronize. To summarize, the Sync_en field should be "0" for this type of operation.

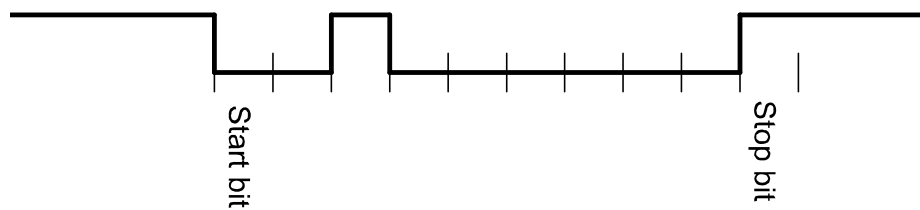


Figure 7. "UART" Code.

Troubleshooting the Data Interface

General Problems

No clock pulses on DATACLK

- If you want to use the DATACLK pin, then make sure the Sync_en field is “1”
- Also remember that DATACLK is always an output from the MICRF transceiver chip

The wanted bit-rate can't be reached

The bit-rate is based on the external crystal and fields in the control word (BitRate_clkS, BitSync_clkS and RefClk_K). Now, the bit-rate generator has a finite resolution — not every bit rate will be “hit”. If an exact bit rate is important, then consider another crystal (note: this will change other settings as well, e.g., the RF frequencies) or do not use the built-in bit synchronizer at all (we do recommend to use it, though). Also note: If both the transmit and receive sides are 505/506, then an inexact hit is acceptable, since the bit rate will be the same for both receive and transmit. If some other RF-chip is used in receiving or transmitting, it might be necessary to make an exact bit rate

The achieved RF frequency and/or bitrate differs from the expected (the calculated).

- Make sure that the crystal frequency of the crystal connected to the MICRF transceiver is used in the calculations. In other words, do not use the crystal connected to the MCU.
- Make sure that the crystal is oscillating at the correct frequency (test capacitive load -- refer to data sheet for both the crystal and the MICRF transceiver)

Transmit Mode Problems

No modulation

- Make sure DATAIXO is an output from the MCU
- VCO modulation: Make sure the modulator is correctly set up
- A,N,M modulation: Make sure A0,N0,M0 holds “tx0” and A1,N1,M1 holds “tx1”

The first bits (elements) are not sent correctly

- VCO mod: Make sure DATAIXO line is tri-stated by the MCU until start-of-modulation. Set DATAIXO as output from MCU immediately before starting to modulate.
- Make sure not to start the modulation until the power amplifier (PA) is completely turned on (use the LD pin or a minimum delay after the PA is turned on).

The very last bit is not sent correctly

Make sure to stay in tx-mode with PA on until the last bit is completely transmitted. Typically, the last bit is set on the DATAIXO line at a falling edge of DATACLK. Then, at the next rising edge, the MICRF transceiver samples the DATAIXO line. Now, make sure not to exit tx-mode until the next rising edge (then the last bit is completely transmitted).

Several equal bits-in-a-row are not sent correctly

VCO modulation: Must use a DC-free code. That is: There is a maximum number of equal bits-in-a-row (or, more precisely, elements-in-a-row) – and the external loop-filter is designed for this. For example, with Manchester Code, the loop filter must be designed to allow two equal elements in-a-row. If another coding is used that increases the number of equal elements in-a-row, then either: A slower loop-filter must be used, or the element rate must be increased (which will limit the maximum “high” or “low” time)

Receive Mode Problems

DATAIXO line is stuck

- Make sure DATAIXO is an input to the MCU (i.e., MCU is not driving DATAIXO low or high).
- If A,N,M modulation: Make sure that the receive frequency is in-the-middle-of the “tx0” and “tx1” frequencies. If the receive frequency equals “tx0” or “tx1”, then it might look like the DATAIXO line is constantly high or low.

DATAIXO gives out “something”, even if no transmitter is present

This is normal behavior and should be expected. The RF chip continuously demodulates the RF (signal or noise) and changes the DATAIXO line – it is important to separate “noise” from “information.”

Clock recovery seems to work, but the first rxed bits of a frame are not OK

Make sure a preamble of 1010... is used when transmitting the frame. Recommended number of transitions is 24 (number of “over-the-air” bits = 24).

In A,N,M mod, the rx-frequency seems to be the one stored in “A1,N1,M1”, not in “A0,N0,M0”

This can happen when going from transmit to receive, and the last data bit clocked into the RF chip is “1” (i.e., the state of DATAIXO at the very last clock-pulse before a load-pulse is made). Make sure the last bit clocked in before making a load-pulse is “0” or “tristate”. Or simply enter the rx-frequency into both A0,N0,M0 and A1,N1,M1.

Conclusion

This app note has shown how to use the DATA interface and how to make an appropriate coding. The user should observe that several other coding methods exist, and selecting a method must be based upon the specific application.

It is the application's requirements that dictate how to handle trade-offs and make selections, such as:

- Bit-banging vs. using a MCU serial module
- Type of line encoding, modulation type and bit rate
- Using/not using Sync_en

MICREL, INC. 2180 FORTUNE DRIVE SAN JOSE, CA 95131 USA
TEL +1 (408) 944-0800 FAX +1 (408) 474-1000 WEB <http://www.micrel.com>

The information furnished by Micrel in this data sheet is believed to be accurate and reliable. However, no responsibility is assumed by Micrel for its use. Micrel reserves the right to change circuitry and specifications at any time without notification to the customer.

Micrel Products are not designed or authorized for use as components in life support appliances, devices or systems where malfunction of a product can reasonably be expected to result in personal injury. Life support devices or systems are devices or systems that (a) are intended for surgical implant into the body or (b) support or sustain life, and whose failure to perform can be reasonably expected to result in a significant injury to the user. A Purchaser's use or sale of Micrel Products for use in life support appliances, devices or systems is a Purchaser's own risk and Purchaser agrees to fully indemnify Micrel for any damages resulting from such use or sale.

© 2008 Micrel, Incorporated.