

AT86RF211

.....
User Guide





Section 1

Overview and Purposes of this Software

1.1 Introduction

The frequency registers of the AT86RF211 are 32 bits wide. The two registers F0 (to set data '0' at $F_{CHANNEL} \pm IF1$) and F1 (to set data 1 at $F_{CHANNEL} \pm deviation$) have to be programmed in transmit mode whilst only F2 (to set the local oscillator frequency) has to be programmed in receive mode. The registers set the various dividers, prescalers and VCO settings used in the synthesizer. The synthesizer of the AT86RF211 is a multi-loop type with a very high level of both accuracy (200 Hz) and settling time (200 μ s typ. to swap from one channel to another). The need to achieve this high performance level necessitated a special and innovative design of the chip leading to a more complex calculation of the register values. All the bits in these registers are scrambled and as there is no easy way to re-calculate each bit in relation with the desired frequency, the calculation can not be embedded into the microcontroller.

Instead, Atmel can provide a demo software, which includes the algorithm to generate the right register corresponding to the desired frequency. This software is delivered with the Evaluation Kit and the Development Kit (a.k.a. Thomlink). The registers are generated one by one. This is very convenient for test purposes and allows an easy evaluation of the RF chip.

The frequency generation software described in this documentation is to enable the calculation of all the registers needed by one application: all the channel transmission frequencies and the receiver LO frequency depending on the reference crystal frequency, the channel spacing the number of channels.

The bytes making up the different registers are delivered into a file the format of which can be decided by the user allowing easy down loading and/or compiling of the created library.

This software provides the means to:

- allow the automatic generation of frequency registers taking into accounts the reference crystal frequency,
- generate library of registers, which can be compiled with the customer's application software (directly into the Flash),
- obviate the need for trimming on the board. (The trimmer is no longer required as the reference crystal frequency is taken into account in the algorithm),
- design easily a Frequency Hopping system by generating all the required channels.

It then becomes a software tool to save time and cost when considering production features and tests.

1.2 Installation and Interface the Software

1.2.1 Installing the Software

Several files are delivered:

<i>trx_dll.dll</i> :	the library itself including the functions to be called.
<i>trx_gen.exe</i> :	a VB6 interface for testing purposes.
<i>trx_exe.exe</i> :	a C++ interface for testing purposes.
<i>trx_dll.lib</i> :	library file to be used by C/C++ applications.
<i>trx_dll.exp</i> :	needed by the .exe.
<i>trx_dll.h</i> :	include file for C/C++ applications.
<i>Api_Trx_Dll.bas</i> :	include file for VB application.

All these files must be installed in the same directory. As a first step, *trx_exe.exe* can be launched and used to create output files.

1.2.2 Interface for the Evaluation of the Software

Two versions of the H/M interface are delivered to evaluate the registers' generation. They are similar: one in C++ (*trx_exe.exe*) and one is in VB6 (*trx_gen.exe*).

TRX_EXE.EXE:

The interface gives access to the channel mode of the .dll. The input parameters are:

<i>Reference frequency (Hz)</i>	The measured frequency of the crystal or the trimmed value of the crystal.
<i>Start channel (Hz)</i>	Center frequency of the first channel.
<i>Deviation (Hz)</i>	Frequency deviation used for FSK modulation (for instance: 10000 if ± 10 kHz has to be synthesized).
<i>Intermediate frequency (\pm Hz)</i>	If the application needs a LO higher than the RF channel, then this value must be positive (for instance: 10700000). If the LO is lower than the RF channel, then this value must be negative (for instance: -10700000).
<i>Step/Channel spacing (Hz)</i>	The spacing between the two successive channels to be synthesized (for instance 100000).
<i>Number of channels</i>	For each channel, 2 Tx frequencies and 1 Rx frequency are calculated (to be synthesized).
<i>Name of the output file</i>	A file will be generated containing all the calculated bytes. If the extension is .c or .h or .cpp, a file ready to be compiled is generated. With other extensions, a text file is generated.
<i>Memory type</i>	This character field is used for the library type output file for compiling options (For instance __flash).

Different resolution modes can be chosen:

<i>High resolution</i>	All the 32 bits (4 bytes) of a frequency register are written in the file (resolution is maximum). This is the recommended mode.
<i>25 bits mode</i>	Only 25 bits (4 bytes + 1 bit) are given. A mask to rebuild to the 32 bits is given (resolution is maximum).
<i>P constant</i>	Some parameters are internally set to a constant value and the frequency register length is decreased to 24 bits (3 bytes). The resolution is then decreased and 200 Hz accuracy can not be achieved any more (resolution is minimum).
<i>Tx/Rx separate</i>	Tx and Rx bytes are listed separately in the output file
<i>Little endian/big endian</i>	Output bytes format. Big endian mode is recommended.

Receiver band choice:

This parameter has to be selected carefully in accordance with the application bandwidth.

<i>No Band</i>	The calculation of the LO register is made without optimization in relation with the receiver bandwidth. Normally not to be used.
<i>Narrow Band</i>	The calculation of the LO register is made with optimization in relation with the receiver bandwidth. To be used if a 455 kHz ceramic filter or a narrow band 10.7 MHz ceramic filter is used => Band Width of the receiver less than 60 kHz. The accuracy of the synthesizer is typ. 200 Hz .
<i>Medium Band</i>	The calculation of the LO register is made with optimization in relation with the receiver bandwidth. To be used if a 455 kHz ceramic filter or a narrow band 10.7 MHz ceramic filter is used => Band Width less than 300 kHz. The accuracy of the synthesizer is typ. 400 Hz .
<i>Wide Band</i>	The calculation of the LO register is made with optimization in relation with the receiver bandwidth. To be used if a 455 kHz ceramic filter or a narrow band 10.7 MHz ceramic filter is used => Band Width less than 600 kHz. The accuracy of the synthesizer is typ. 1 kHz with maximum of about 3 kHz.

Figure 1-1. TRX_EXE.EXE Display Interface

The screenshot shows a dialog box titled "TRX Channel". It contains several input fields and a "Generate !" button at the bottom. The parameters are as follows:

Parameter	Value
Reference Frequency (Hz)	10245000
Start Channel (Hz)	915300000
Deviation (+/- Hz)	10000
Intermediate Freq (+/- Hz)	10700000
Step (Hz)	100000
Number of channels	10
Output File	output.c
Memory Type	const
Mode	High Resolution
TX/RX Separate	<input checked="" type="checkbox"/>
25 Bits	<input type="checkbox"/>
Little Endian	<input type="checkbox"/>
P Const	0
Big Endian	<input checked="" type="checkbox"/>
Band	No Band
Medium Band	<input type="checkbox"/>
Narrow Band	<input checked="" type="checkbox"/>
Wide Band	<input type="checkbox"/>

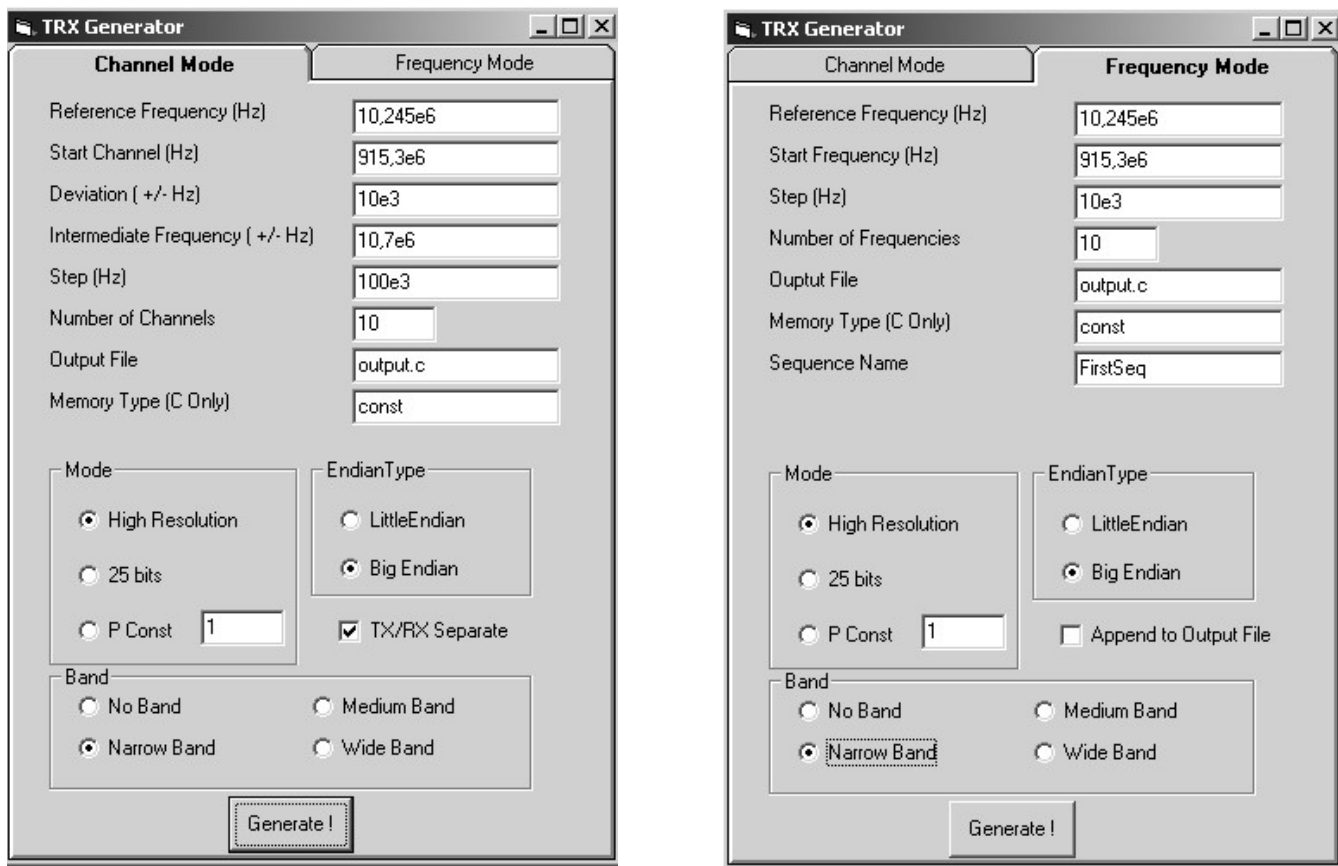
TRX_GEN.EXE:

The interface gives access to the channel mode and to the frequency mode of the .dll. The input parameters for the channel mode are then the same for the TRX_EXE.EXE version. For the frequency mode, the input parameters are the same except:

- The intermediate frequency does not have to be programmed anymore as the user directly indicates the frequencies he wants to generate.
- The step is the spacing between the desired frequencies.

In this mode, the Tx frequencies and the Rx frequencies must be calculated separately.

Figure 1-2. TRX_GEN.EXE Display Interface



1.3 Structure of the Library

The library is: `trx_dll.dll`. It can be used by any kind of VB or C/C++ application software. It is also possible to interface it with high level measurement or production bench software.

It must be included and compiled with the relevant library:

- **trx_dll.h** for applications written with C/C++.
- **Api_Trx_Dll.bas** for application written with Visual Basic.

The `trx_dll.dll` contains the following functions:

DWORD WINAPI	TRXFreqMode():	Frequency range sweep and generation of the relevant registers.
DWORD WINAPI	TRXChannelMode():	Channel range sweep and generation of the relevant register.
void WINAPI	TRXSyntheHz():	Direct call of the function, which calculates a register.

The H/M interface described above is calling these procedures. Thus what is generated by this demo software is exactly what is generated using the `.dll` directly on customer's application.



1.3.1 TRXFreqMode

Function generating the relevant registers covering the specified frequency range.

Prototype of this function:

```
DWORD WINAPI TRXFreqMode ( DOUBLE    dRefFreq
    , DOUBLE    dStartFreq
    , DOUBLE    dStepFreq
    , DWORD     nNbFreq
    , DWORD     nModeGen
    , DWORD     nP
    , DWORD     nBandType
    , char      *szFileName
    , char      *szMemType
    , char      *szSequenceName
    , DWORD     bAppend
);
```

Description of the parameters:

- **dRefFreq**
Reference frequency (crystal) of the RF board (Hz)
- **dStartFreq**
Start frequency of the range to be synthesized (Hz)
- **dStepFreq**
Frequency step (Hz)
- **nNbFreq**
Number of frequencies to be synthesized
- **nModeGen**
Generation mode of the files. This parameter allows to specify the way the calculated registers will be written into the output file:
 - 0** High resolution mode: registers are written completely in the output file (4 bytes per register thus 32 bits)
 - 1** 25 bits mode: 7 bits into the registers are constant, thereafter it is possible to compact them to 25 bits; a set of n registers (n*4 bytes in the high resolution mode) is then compacted into (n*3 + (n+7)/8) bytes.
 - 2** P constant mode: the calculation of the registers is made with a constant value for P, thereby 24 bits (3 bytes per register).

The registers are written as Little Endian (LS Bytes first) but it is possible to have them written as Big Endian (MS Bytes first) by selecting the MSB of this parameter nModeGen (i.e. 32nd bit) to 1.
- **nP**
P value in the case where Constant P mode is used (between 0 and 6).
- **nBandType**
Choice of the LO frequency accuracy for sensitivity optimization:
 - 0** No Optimization: not to be used

- 1 Narrow Band: optimizes sensitivity for applications having a bandwidth in reception of less than 60 kHz.
- 2 Medium Band: optimizes sensitivity for applications having a bandwidth in reception of less than 300 kHz.
- 3 Wide Band: optimizes sensitivity for applications having a wider bandwidth.

■ **szFileName**

Character (Ansi C format) describing the name of the file in which are written the registers. If the format is set with .h, .c ou .cpp then the .dll generates a C format file otherwise CSV format is generated.

■ **szMemType**

Character (Ansi C format) describing a specific word to be added before the declaration of the C structure filled with the data. This parameter is used only if the DLL generates a C format output file. Example: " const "

■ **szSequenceName**

Character (Ansi C format) describing the name of the sequence; if the output is a C format, the generated C structure will have this name.

■ **bAppend**

Boolean describing if data have to be added at the end of the file (value <> 0) or if the new calculation erase the previous and renew the entire file (value = 0): this allows the generation of several frequency ranges.

Return of the function:

- 0 Function was correctly running.
- 1 Memory allocation error occurred.
- 2 Output file access failed.
- 3 Invalid parameter value: Wrong mode, number of channels is null, P value is out of range.
- 4 In the 25 bits mode, the range of channels to be synthesized does not allow all the 7 bits to remain constant.

1.3.2 TRXChannelMode

Function generating the relevant registers covering the specified channels range.

Prototype of this function:

```
DWORD WINAPI TRXChannelMode ( DOUBLE    dRefFreq
    , DOUBLE    dStartChannel
    , DOUBLE    dDevTx
    , DOUBLE    dInterFreq
    , DOUBLE    dChannelStep
    , DWORD     nNbChannel
    , DWORD     nModeGen
    , DWORD     nP
    , DWORD     nBandType
    , char      *szFileName
    , char      *szMemType
    , DWORD     bSeparateTxRx
);
```



Description of the parameters:

■ dRefFreq

Reference frequency (crystal) of the RF board (Hz).

■ dStartChannelStart frequency channel to be synthesized (Hz).

■ dDevTx

Deviation for the transmission (Hz). The dll synthesizes two frequencies for the transmission: $F0 = FChannel - dDevTx$, $F1 = FChannel + dDevTx$.

■ dInterFreq

Intermediate frequency for the reception (Hz). The .dll synthesizes one frequency for the reception: $F = FChannel + dInterFreq$ (LO frequency).

■ dChannelStep

Frequency step between two adjacent channels (Hz).

■ nNbChannel

Number of channels for which the .dll is going to synthesize the two frequencies for transmission and the frequency for the reception.

■ nModeGen

Generation mode of the files. This parameter allows to specify the way the calculated registers will be written into the output file:

0 High resolution mode: registers are written completely in the output file (4 bytes per register thus 32 bits).

1 25 bits mode: 7 bits into the registers are constant, thereafter it is possible to compact them to 25 bits ; a set of n registers ($n \times 4$ bytes in the high resolution mode) is then compacted into $(n \times 3 + (n+7)/8)$ bytes.

2 P constant mode: the calculation of the registers is made with a constant value for P, thereby 24 bits (3 bytes per register).

The registers are written as Little Endian (LS Bytes first) but it is possible to have them written as Big Endian (MS Bytes first) by selecting the MSB of this parameter nModeGen (i.e. 32nd bit) to 1.

■ nP

value in the case where constant P mode is used (between 0 and 6).

■ nBandType

Choice of the LO frequency accuracy for sensitivity optimization:

0 No Optimization: not to be used.

1 Narrow Band: optimizes sensitivity for applications having a bandwidth in reception of less than 60 kHz.

2 Medium Band: optimizes sensitivity for applications having a bandwidth in reception of less than 300 kHz.

3 Wide Band: optimizes sensitivity for applications having a wider bandwidth.

■ szFileName

Character (Ansi C format) describing the name of the file in which are written the registers. If the format is set with ".h", ".c" or ".cpp" then the .dll generates a C format file otherwise CSV format is generated.

■ szMemType

Character (Ansi C format) describing a specific word to be added before the declaration of the C structure filled in with the data. This parameter is used only if the .dll generates a C format output file . Example: " const ".

■ bSeparateTxRx

Boolean describing if the Tx and Rx frequencies have to be separated (value $\neq 0$) or if they are set by channels (value = 0). In the first case (frequencies are separated), the .dll first writes a structure for the Tx frequencies (FTx0, FTx1 channel 1 ; FTx0, FTx1 channel2 ; ... ; FTx0, FTx1 channel n ; ...) then a second structure for the Rx frequencies (FRx channel 0 ; FRx channel 1 ; ... FRx channel n ; ...). In the second case, the .dll writes one structure in which the frequencies are set by channel (FTx0, FTx1, FRx channel 0 ; FTx0, FTx1, FRx channel 1 ; ... FTx0, FTx1, FRx channel n ; ...).

Note: In the 25 bits mode, Tx and Rx registers must be separated in the output file (bSeparateTXRX = TRUE) ;

Return of the function:

- 0 Function was correctly running.
- 1 Memory allocation error occurred.
- 2 Output file access failed.
- 3 Invalid parameter value: Wrong mode, number of channels is null, P value is out of range, Tx/Rx registers are not separated in 25 bits mode.
- 4 In the 25 bits mode, the range of channels to be synthesized does not allow all the 7 bits to remain constant.

1.3.3 TRXSyntheHz

Low level function calculating the registers directly.

Prototype of the function:

```
void WINAPI TRXSyntheHz ( DWORD FSKmode
    , DOUBLE dRefFreq
    , DOUBLE dChannelFreq
    , DOUBLE dInterFreq
    , DOUBLE dDevTx
    , DWORD nP
    , DWORD nBandType
    , DWORD *pnReg1
    , DWORD *pnReg2
);
```

Description of the parameters:

■ FSKmode

0 Calculation for the reception mode (in this case, dDevTx must be null).

1 Calculation for the transmission mode (in this case, dInterFreq must be null).

■ dRefFreq

Reference frequency (crystal) of the TRX board (Hz).

■ dStartChannel

Channel frequency (Hz).

■ dInterFreq

Intermediate frequency for the reception (Hz). The .dll synthesizes one frequency for the reception: $F = F_{\text{Channel}} + d_{\text{InterFreq}}$ (LO frequency).

■ dDevTx

Deviation for the transmission (Hz). The .dll synthesizes two frequencies for the transmission: $F_0 = F_{\text{Channel}} - d_{\text{DevTx}}$, $F_1 = F_{\text{Channel}} + d_{\text{DevTx}}$.

■ nP

P value. If -1, the .dll makes the calculation with the best value, otherwise the calculation is made with the given parameter (between 0 and 6).

■ pnReg1

Pointer on a 32 bits word in which the .dll is writing the value of the F0 register in transmission mode or F2 in the reception mode.

■ pnReg2

Pointer on a 32 bits word in which the .dll is writing the value of the F1 register in transmission mode; unused for the reception mode.

■ nBandType

Choice of the LO frequency accuracy for sensitivity optimization:

0 No Optimization: not to be used.

1 Narrow Band: optimizes sensitivity for applications having a bandwidth in reception of less than 60 kHz.

2 Medium Band: optimizes sensitivity for applications having a bandwidth in reception of less than 300 kHz.

3 Wide Band: optimizes sensitivity for applications having a wider bandwidth.

Return of the function:

Note: if a parameter is wrong (ex: wrong value for P), the .dll does make no calculation (null is written in all the word pointed).

Section 2

Outputs and Files' Organization

-
- 2.1 Introduction** Three kinds of files are generated:
- .c,.h,.cpp: They can be considered as library. Directly compilation on a production bench is possible to allow the down loading in the Flash of the micro-controller of the RF application board.
 - .log: They contain frequencies' information for verification purposes.
 - text files: they can be read automatically or manually by any tools and registered as frequency registers.
- All these files are made of 2 fields:
- Comments field: It recalls all the information the user has specified for the calculation.
 - Useable field: It contains the relevant register information that is of interest for the application.
-
- 2.2 Library Type File (.c, .h, .cpp)** Useable field:
- A structure (TRXReg) is created and contains the registers, which are calculated. If Tx and Rx are specified to be separated, then 2 structures are created (TRXTxReg and TRXRxReg).

Figure 2-1.

```

/*
TRX Dll version : V1.0.1
File generate : Thursday, March 07, 2002 17:13:17
Function TRXChannelMode
Param :
    Reference Frequency : 10245000 Hz
    Start Channel       : 915300000 Hz
    Deviation (TX)      : 10000 Hz
    Intermediary Freq(RX) : 10700000 Hz
    Step                : 100000 Hz
    Number of channels  : 1
*/
const struct {
    unsigned short nNbRegs;
    unsigned char  cMode;
    unsigned char  bBigEndian;
    unsigned char  ArRegs[12];
} TRXReg = { 3, 0, 1, {
0xb1, 0x9, 0xc6, 0x4e, 0xc, 0x8, 0x46, 0x6a, 0x95, 0xb1, 0x42, 0x7c
}};

```

Annotations in the image:

- An arrow points from the text "character field" to the closing comment tag `*/`.
- An arrow points from the text "1 channel to be generated" to the value `1` in the "Number of channels" parameter.
- Three arrows point from labels below to specific registers in the `ArRegs` array:
 - `(Tx - Dev) register` points to `0xb1`.
 - `(Tx + Dev) register` points to `0xc`.
 - `Rx (LO) register` points to `0x95`.

2.3 Log Type File (.log)

Whatever the file option, this file is created. It gives the generated frequencies and the difference between the desired frequency and the synthesized frequency.

Figure 2-2.

```

/*
TRX Dll version : V1.0.1
File generate : Thursday, March 07, 2002 18:03:49
Function TRXChannelMode
Param :
    Reference Frequency : 10245000 Hz
    Start Channel       : 915300000 Hz
    Deviation (TX)      : 10000 Hz
    Intermediary Freq(RX) : 10700000 Hz
    Step                : 100000 Hz
    Number of channels  : 1
*/
Channel :
    Tx Freq0 Input : 915290000    Output : 915290011    Delta : -11
    Tx Freq1 Input : 915310000    Output : 915309868    Delta : 131
    Rx Freq  Input : 926000000    Output : 926000038    Delta : -38

```

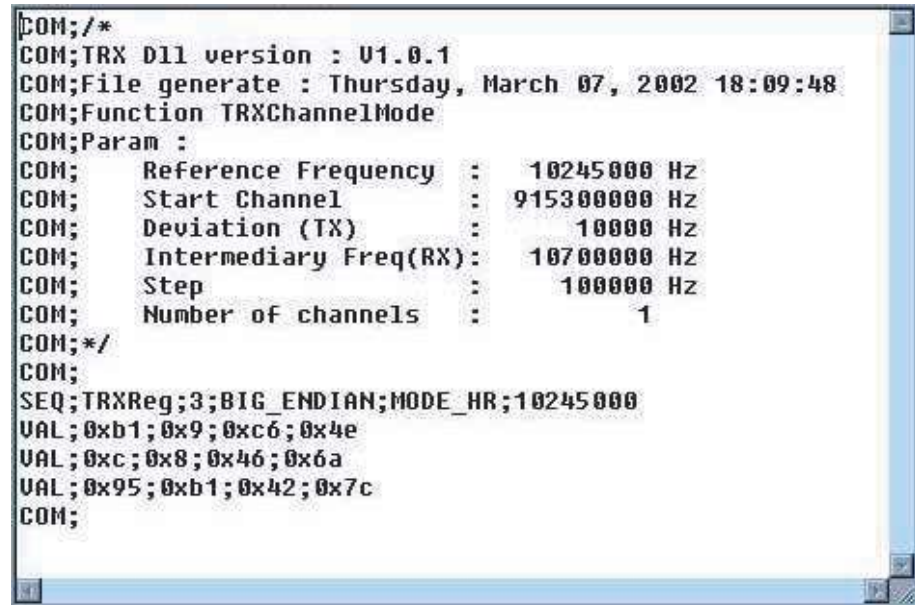
Annotations in the image:

- An arrow points from the text "desired frequencies" to the "Input" column of the frequency data.
- An arrow points from the text "synthesized frequencies" to the "Output" column of the frequency data.

2.4 Text type file

It contains comments (COM;), structure of the registers (SEQ;) and the bytes of the registers (VAL;).

Figure 2-3.



```
COM; /*
COM; TRX Dll version : V1.0.1
COM; File generate : Thursday, March 07, 2002 18:09:48
COM; Function TRXChannelMode
COM; Param :
COM;   Reference Frequency   : 10245000 Hz
COM;   Start Channel        : 915300000 Hz
COM;   Deviation (TX)       : 10000 Hz
COM;   Intermediary Freq(RX): 10700000 Hz
COM;   Step                  : 100000 Hz
COM;   Number of channels   : 1
COM; */
COM;
SEQ; TRXReg;3;BIG_ENDIAN;MODE_HR;10245000
VAL; 0xb1;0x9;0xc6;0x4e
VAL; 0xc;0x8;0x46;0x6a
VAL; 0x95;0xb1;0x42;0x7c
COM;
```

This file is of interest for applications, which need to read the generated registers easily. It is automatically selected if .c, .h, .cpp are not specified as input parameters.

Section 3

High Resolution/Low Resolution Modes

The high resolution mode for the frequency registers must be understood as the 32 bits mode. It allows an accuracy of about 200 Hz concerning the frequency, which is synthesized. **This mode is recommended as it offers both high accuracy and easy handling of the registers (no scrambling/descrambling).**

The 25 bits mode delivers the registers coded with 25 bits instead of 32 bits. 5 bits are constant whatever the generated frequency and 2 bits only depend on the chosen frequency band. It allows storing less bytes in the memory of the microcontroller but it is necessary to unscramble the stored bytes. The high resolution of the synthesizer is kept and the frequencies are alike the ones generated with the high resolution mode (need for an unscramble operation in the microcontroller).

The P=cst mode offers the ability to have only 24 bits for the frequency registers. To do so, 3 bits remain constant and the resolution of the synthesizer reduces. It can be expected to have an accuracy of 10 kHz instead of 200 Hz. This mode is not recommended. Only if very stringent memory allocation/size and very low resolution needs are required, can it be used. P must remain between 0 and 6.

Only the 25 bits mode and the P=cst mode will be detailed. The high resolution mode does not have specific features and has already been described.

3.1 25 bits mode

In the 32 bits frequency register, the bits 0, 8, 14, 17 and 25 are constant whatever the generated frequency.

■ 0, 8, 17, 25 = '0'.

■ 14 = '1'.

The bits 11 and 18 depend on the frequency band, which is chosen.

Generated frequency	400-450 MHz	450-480 MHz	800-850 MHz	900-950 MHz
Bit 11	1	1	0	0
Bit 18	1	0	1	0

Notes: in this mode, Tx/Rx must be chosen separated as the Tx and Rx frequencies can be into 2 different columns.

3.1.1 Output Structure **szMemType** struct {

```

    unsigned short    nNbRegs;
    unsigned short    iChangeMask;
    unsigned char     cMode;
    unsigned char     bBigEndian;
    unsigned char     filler[2];
    unsigned char     ArMask0[4];
    unsigned char     ArMask1[4];
    unsigned char     ArRegs[Dim];
} SeqName = ...

```

Description of the parameters:

<i>SzMemType</i>	As specified when calling the function
<i>nNbRegs</i>	number of generated registers
<i>iChangeMask</i>	Index of the register from which mask2 has to be used (0 if no change in the sequence)
<i>cMode</i>	Sequence mode => 1 for a 25 bits mode sequence
<i>bBigEndian</i>	Boolean indicating the order of the bytes (big endian 1 as recommended)
<i>filler</i>	Filling field to allow alignment in the presentation of the file
<i>ArMask0</i>	32 bits (4 bytes) mask to apply to the registers after uncompacting - this mask only applies to registers with index strictly below the index <i>iChangeMask</i>
<i>ArMask1</i>	32 bits (4 bytes) mask to apply to the registers after uncompacting - this mask only applies to registers with index equal or above the index <i>iChangeMask</i>
<i>ArRegs</i>	Table containing the registers - they are set 8 by 8 and written: 1 bytes with all the 25 th bits + 8x3 bytes containing the registers.
<i>Dim</i>	Dimension of the table of bytes => NbReg * 3 + ((NbReg+7)/8)

3.1.2 Description of the Compacting

Considering one frequency band, only 25 bits have to be stored. The software is compacting the 32 bit register into a 25 bit register: the 7 constant bits are removed and replace by the 7 last ones (MSB), and the bit n°24 (the 25th) of all the registers are stored into dedicated bytes.

Bits are changed as expressed below:

- Bit n° 0 <= bit n° 26
- Bit n° 8 <= bit n° 27
- Bit n° 11 <= bit n° 28
- Bit n° 14 <= bit n° 29
- Bit n° 17 <= bit n° 30
- Bit n° 18 <= bit n° 31

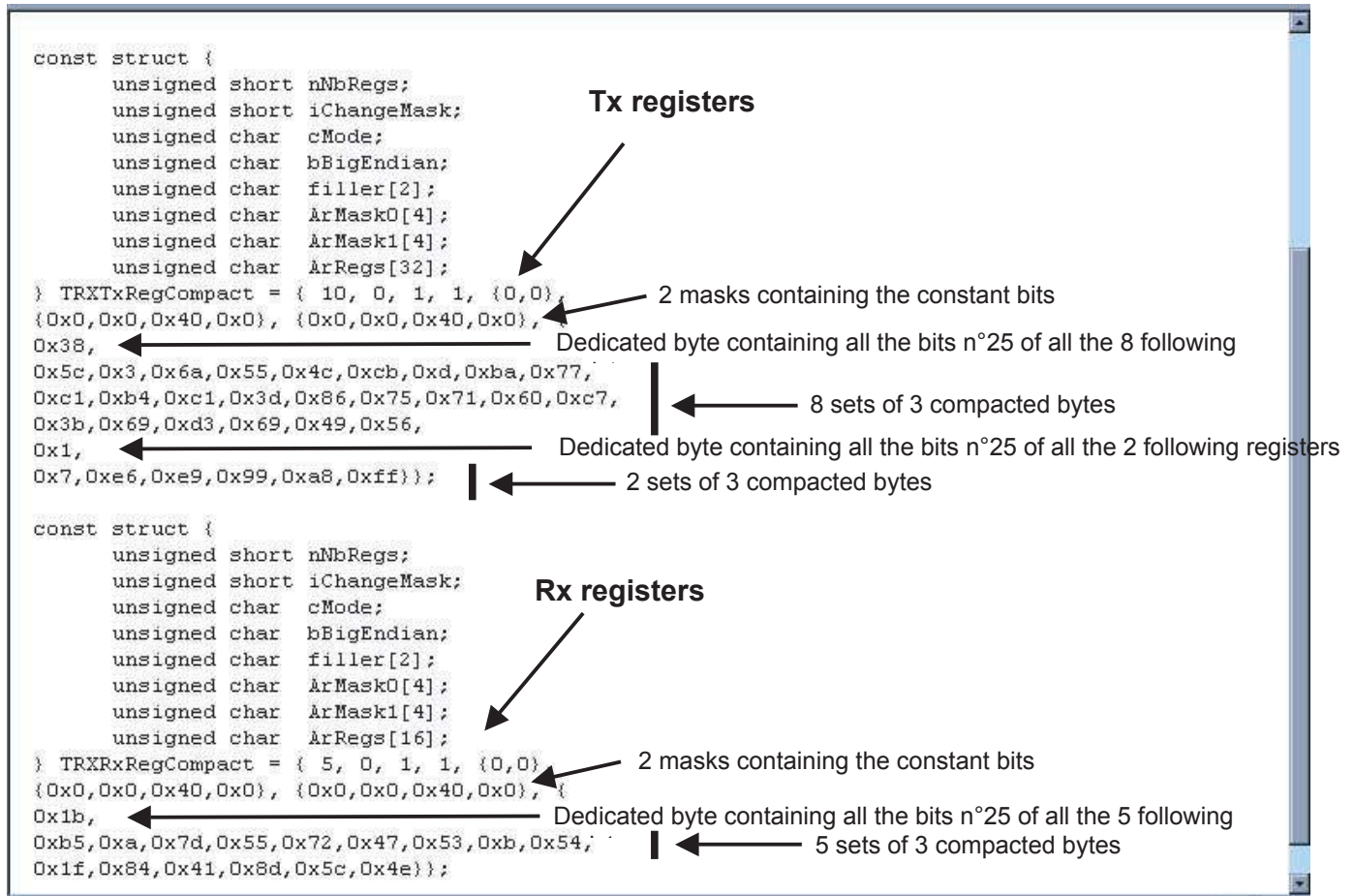


Then, the register size is only 25 bits. To allow only 24 bits (3 bytes), the 25th is stored into the dedicated byte. Thereafter, only 3 bytes are necessary to store one frequency register.

- Notes:
1. the dedicated byte must be renewed every 8 registers thus for a set of $8 \times 3 = 24$ bytes (8 frequencies),
 2. the dedicated byte is filled from the LSB up to the MSB.

The example below shows the generated file for 5 channels in the 915 MHz band. There are 10 registers (2x5 frequencies) for the Tx and 5 registers (1x5 registers) for the Rx.

Figure 3-1.



The compacted registers of 24 bits are given thanks to 3 bytes:

- 0x5c, 0x3, 0x6a are the 3 compacted bytes of the 1st Tx register.
- The 25th bit of this 1st Tx register is '0' (LSB of 0x38).
- 0xb5, 0xa, 0x7d are the 3 compacted bytes of the 1st Rx register.
- The 25th bit of this 1st Rx register is '1' (LSB of 0x1b).

To uncompact the registers, the reverse operation has to be done. The 3 bytes are written as LSB into a 32 bits register. The bits n°0, 8, 11, 14, 17 and 18 are copied as expressed below:



- Bit n° 26 <= bit n° 0
- Bit n° 27 <= bit n° 8
- Bit n° 28 <= bit n° 11
- Bit n° 29 <= bit n° 14
- Bit n° 30 <= bit n° 17
- Bit n° 31 <= bit n° 18

then bits n°0, 8, 11, 14, 17, 18 and 25 are set to '0'. Therefore, the 32 bits register only needs to retrieve the constant bits: this is done applying the mask given in the generated file. A OR operation is done between the mask and the register.

The example given above is then uncompact as:

- **0x5c, 0x3, 0x6a** with LSB of **0x38** for the 25th bits => 0x88, 0x58, 0x2, 0x6a.
- (0x88, 0x58, 0x2, 0x6a) OR (0x0, 0x0, 0x40, 0x0) => **0x88, 0x58, 0x42, 0x6a** is the register to be programmed.

- Notes:
1. the generated file gives a mask to retrieve the values of the bits.
 2. in the example, the mask is {0x00, 0x00, 0x40, 0x00} because only the bit 14 has to be set to '1'.
 3. it is possible to have 2 masks for the same file. The normal operation only uses one, but 2 masks could be necessary if the frequencies to be generated are in 2 different ranges (for instance, Rx frequencies between 896.3 and 904.3 MHz for channels between 907 and 915 MHz). If 2 masks are used, the iChangeMask indicates where the change occurs.
 4. if the text file output is chosen, in addition to COM;, SEQ;, VAL; fields, there is a MSK; field which delivers the mask value. The first VAL; field gives the dedicated byte for all the 25th bits.

3.2 P = cst mode

In the P=cst mode, the P divider ratio of the PLL is kept constant. With the addition of the constant bits of the 25 bits mode, it is possible to code the register with only 3 bytes and no dedicated byte (as for the 25th bit of the 25 bits mode).

- bits n°0, n°8, n°17, n°25 = '0'.
- bit n°14 = '1'.
- bits n°3, n°12, n°21 depending on the value of P. **P=3 is recommended then bit 3 = '1', bit 12= '1', bit 21 = '0'** (these values are taken into account in the mask).

Note: In this mode, bit n°11 and bit n°18 can vary, then no need for a dedicated byte in the file structure. The 3 bytes are directly available.

3.2.1 Output Structure

```
szMemType struct {
    unsigned short nNbRegs;
    unsigned char  cMode;
    unsigned char  bBigEndian;
    unsigned char  ArMask[4];
    unsigned char  ArRegs[Dim];
} SeqName = ...
```

Description of the parameters:

<i>SzMemType</i>	As specified when calling the function
<i>nNbRegs</i>	Number of generated registers
<i>cMode</i>	Sequence mode => 2 for a P=cst mode sequence
<i>bBigEndian</i>	Boolean indicating the order of the bytes (big endian 1 as recommended)
<i>ArMask</i>	32 bits (4 bytes) mask to apply to the registers after uncompacting
<i>ArRegs</i>	Table containing the registers - they are set 8 by 8 and written: 1 byte with all the 25 th bits + 8x3 bytes containing the registers.
<i>Dim</i>	Dimension of the table of bytes => NbReg * 3

3.2.2 Description of the Compacting

Considering one frequency band, only 24 bits have to be stored. The software is compacting a 32 bit register into a 24 bit register: the 8 constant bits are removed and replace by the 7 last ones.

Bits are changed as expressed below:

- Bit n° 0 <= bit n° 24
- Bit n° 3 <= bit n° 26
- Bit n° 8 <= bit n° 27
- Bit n° 12 <= bit n° 28
- Bit n° 14 <= bit n° 29
- Bit n° 17 <= bit n° 30
- Bit n° 21 <= bit n° 31

Then, the registers' size is only 24 bits (from n°0 to n°23). Thereafter, only 3 bytes are necessary to store one frequency register.

The example below shows the generated file for 5 channels in the 915 MHz band. There are 10 registers (2x5 frequencies) of 3 bytes for the Tx and 5 registers (1x5 registers) of 3 bytes for the Rx.

Figure 3-2.

```

FUNCTION TRXChannelMode
Param :
    Reference Frequency : 10245000 Hz
    Start Channel       : 915300000 Hz
    Deviation (TX)      : 10000 Hz
    Intermediary Freq(RX): 10700000 Hz
    Step                : 100000 Hz
    Number of channels  : 5
    P                   : 2
*/

const struct {
    unsigned short nNbRegs;
    unsigned char  cMode;
    unsigned char  bBigEndian;
    unsigned char  ArMask[4];
    unsigned char  ArRegs[30];
} TRXTxRegCompact = { 10, 2, 1,
{0x0,0x0,0x50,0x0},
0xd1,0x44,0xcf,0xb0,0x81,0xe2,0x33,0x45,0x5f,
0x6b,0xd3,0xf1,0xa9,0x54,0xc6,0x1,0xc6,0x46,
0xd1,0xf5,0xdf,0x69,0x63,0x5b,0xa3,0xe1,0x79,
0xd3,0x2,0x44}};

const struct {
    unsigned short nNbRegs;
    unsigned char  cMode;
    unsigned char  bBigEndian;
    unsigned char  ArMask[4];
    unsigned char  ArRegs[15];
} TRXRxRegCompact = { 5, 2, 1,
{0x0,0x0,0x50,0x0},
0xb,0x25,0xdc,0x71,0x62,0x4f,0xe1,0x57,0xdf,
0xa1,0x42,0x46,0xa9,0xd5,0xd7}};

```

Tx registers

1 mask containing the constant bits (P= 2 in this example)

10 sets of 3 compacted bytes

Rx registers

1 mask containing the constant bits

5 sets of 3 compacted bytes

The compacted registers of 24 bits are given thanks to 3 bytes:

- 0xd1, 0x44, 0xcf are the 3 compacted bytes of the 1st Tx register.
- 0x0b, 0x25, 0xdc are the 3 compacted bytes of the 1st Rx register.

To uncompact the registers, the reverse operation has to be done. The 3 bytes are written as LSB into a 32 bits register. The bits n° 0, 3, 8, 12, 14, 17 and 21 are copied as expressed below:

- Bit n° 24 <= bit n° 0
- Bit n° 26 <= bit n° 3
- Bit n° 27 <= bit n° 8
- Bit n° 28 <= bit n° 12
- Bit n° 29 <= bit n° 14
- Bit n° 30 <= bit n° 17
- Bit n° 31 <= bit n° 21

then bits n° 0, 3, 8, 12, 14, 17, 21 and 25 are set to '0'. Therefore, the 32 bits register only needs to retrieve the constant bits: this is done applying the mask given in the generated file. An OR operation is done between the mask and the register bit to bit.

The example given above is then uncompact as:

- 0xd1, 0x44, 0xcf => 0x25, 0xd1, 0x4, 0xc6
- (0x25, 0xd1, 0x4, 0xc6) OR (0x0, 0x0, 0x50, 0x0) => (0x25, 0xd1, 0x54, 0xc6) is the register to be programmed.

- Notes:
1. The generated file gives a mask to retrieve the values of the bits.
 2. In the example, the mask is {0x00, 0x00, 0x50, 0x00} because the bits n°14 and n°12 have to be set to '1'.
 3. If the text file output is chosen, in addition to COM;,, SEQ;,, VAL; fields, there is a MSK; field which delivers the mask value. The first VAL; field gives the dedicated byte for all the 25th bits.



Atmel Headquarters

Corporate Headquarters

2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 487-2600

Europe

Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia

Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan

9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Memory

2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 436-4314

Microcontrollers

2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards

Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

RF/Automotive

Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/ High Speed Converters/RF Datacom

Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-76-58-30-00
FAX (33) 4-76-58-34-80

e-mail

literature@atmel.com

Web Site

<http://www.atmel.com>

© Atmel Corporation 2002.

Atmel Corporation makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in Atmel's Terms and Conditions located on the Company's web site. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of Atmel are granted by the Company in connection with the sale of Atmel products, expressly or by implication. Atmel's products are not authorized for use as critical components in life support devices or systems.

ATMEL® is the registered trademark of Atmel.

Other terms and product names may be the trademarks of others.



Printed on recycled paper.