

Implementation of a WCDMA Rake Receiver on a TMS320C62x™ DSP Device

Wireless ASP Products

ABSTRACT

A number of techniques can be used to search for the paths in a wideband code division multiple access (WCDMA) digital signal processor (DSP) radio. Overall, the millions of instructions per second (MIPS) required to do this search are approximately 2% of the brute force approach used in a code division multiple access (CDMA) search of application-specific integrated circuit (ASIC) devices. The search is performed at a level of power where a DSP-based rake receiver can be implemented and reception can be improved. This 100-MHz engine searches for paths and receives data using a 6-finger rake receiver at 58 MHz, thereby implementing a complete rake receiver on a single TMS320C62x™ DSP device.

Contents

1	CDMA Theory	3
	1.1 Spreading and Scrambling	3
	1.2 Multipath Signals	3
2	The Basic System	4
	2.1 Control and Data Channels	4
	2.2 Rake Energy Correlations	4
	2.3 Path Selection	5
	2.3.1 Tracking	5
	2.3.2 Training	6
	2.4 Rake Decoding	7
	2.5 Radio Feedback	7
3	Implementation	8
	3.1 Selection of Rake Receiver Simulation With the TMS320C62x™ DSP EVM	9
	3.2 TMS320C62x™ DSP EVM Memory Mapping	9
	3.3 Buffering Scheme	9
	3.4 TMS320C62x™ DSP EVM Usage	10
	3.5 Control File Control.Rake Structure	10
	3.6 Control.cpp (Generate on PC)	12
	3.7 Control.c (Receive on TMS320C62x™ DSP)	13
	3.8 Rake.c	13
	3.8.1 Rake_Reset (global)	14
	3.8.2 Rake_Energy (local)	14
	3.8.3 Rake_Energy_Update (local)	14
	3.8.4 Rake_Search (global)	15
	3.8.5 Rake_Srch_1 (global)	15
	3.8.6 Rake_Receive (global)	16

3.9	Scramble.c	17
3.9.1	Scramble_Init (global)	18
3.9.2	Scramble (global)	18
3.9.3	Implementation of Scrambling Code Generation	19
3.10	Turbodec.c	22
3.11	Turboenc.c	23
3.12	Turbo.h	23
3.13	Umts.h	23
4	Algorithm Performance	23
5	Loading on the TMS320C62x™ DSP Device	25
6	Conclusions	26

List of Figures

1	Phase Rotation	4
2	Path Search Sequence	6
3	Path Probability Tracking	6
4	Flowchart of Rake Receiver Simulation and Communication Between the PC and the TMS320C62x™ DSP EVM	8
5	Buffering of Rake Receiver Input Data	10
6	Initialization and Beginning of Consecutive x-Sequence Generation	20
7	Initialization and Beginning of Consecutive y-Sequence Generation	21
8	Monte Carlo Model	24
9	Turbo Decoding Error Correction	24
10	ADC Errors Versus Resolution	25
11	Real-Time CPU Load for a 200-MHz TMS320C62x™ DSP Device	26

List of Tables

1	CDMA Orthogonal Codes	3
2	TMS320C62x™ EVM—PC Communications With HPI Common Memory Map: Data Changes for Each Slot	9
3	Receive Error Rates	24
4	DSP Cycle Load	25
5	DSP Memory Load	26

1 CDMA Theory

The system described in this application report is based on an early version of the European Telecommunications Standards Institute (ETSI) Universal Mobile Telecommunications Systems (UMTS) standard. Although the concepts remain the same, some of the numbers are different.

1.1 Spreading and Scrambling

In a CDMA system, channels are broadcast on the same frequency using orthogonal spreading codes or patterns. The orthogonal nature of these patterns means that when a reference pattern is correlated with a received pattern, the result is 0 for all other signals that are not required. For the desired signal, the result is non-zero, with the sign ultimately giving the value of the transmitted bit, that is 0 or 1. Table 1 shows the results of multiplying each of the orthogonal vectors with each of the reference vectors for a spreading factor of 4. A spreading factor of 4 means that each bit that is sent and received is represented by 4 chips.

Table 1. CDMA Orthogonal Codes

	(1,1,1,1)	(1,1,-1,-1)	(1,-1,1,-1)	(1,-1,-1,1)
(1,1,1,1)	4	0	0	0
(1,1,-1,-1)	0	4	0	0
(1,-1,1,-1)	0	0	4	0
(1,-1,-1,1)	0	0	0	4

To obtain a unity gain system, the result is divided by the spreading factor, which is 4, as shown in Table 1. In the real system, spreading factors as large as 256 are deployed.

Implementing only the spreading codes however is not enough. Long runs of 1s or (-1s) can be produced and this affects both clock recovery and transmitted power levels. Also, if an adjoining cell uses the same spreading pattern, there could be a conflict. To avoid both these problems, the data values are scrambled with a known pseudo-random scrambling code that both separates adjoining cells and removes the long runs. This scrambling code is always different between adjoining cells. Also, if the maximum delay path (delay spread) is greater than a bit period, the receiver has a better chance of determining bit synchronization by using the spreading and scrambling codes together.

1.2 Multipath Signals

Mobile radio environments require that both the mobile station transceiver (MT) and base station transceiver (BTS) maintain a high-quality link, regardless of the position of the MT within the cell. You cannot assume that the aeriels for both the MT and the BTS are positioned correctly to eliminate multipath signals. Therefore, for narrow-band systems, where there are a small number of strong multipath signals at the receiver falling within a symbol period, software-based channel equalization has been used to correct inter-symbol interference (ISI).

Due to the wideband nature of CDMA systems (that is, high chip rates), these paths can be more than one CDMA bit (chip) wide and as such, traditional equalization is no longer an option. Instead, a technique is needed that receives all the paths and then combines them. A rake receiver is a class of receiver that receives signals on as many multipaths as possible. The rake receiver combines the signals from all of these paths to produce one clear signal that is stronger than the individual components. Individual paths are found (synchronized to) by cross-correlating a reference pattern with the received signal.

2 The Basic System

The basic WCDMA DSP radio consists of control and data channels, rake energy correlations, path selection, rake decoding, and radio feedback.

2.1 Control and Data Channels

The data and control channels are transmitted on the in-phase (I) and quadrature (Q) components of the radio system respectively ($D_I + jD_Q$). These signals are scrambled by a complex scrambling code $C_I + jC_Q$, by mixing, producing $C_I D_I + jC_Q D_Q$. All other components are zero because there is no Q component in an I signal, nor is there an I component in a Q signal. In this standard, a known data pattern is transmitted on the first 6 bits of the control channel. By mixing the reference signal, scrambling code, and the known data pattern, a much longer reference correlation model can be obtained. This pattern is then used to search for a particular channel. This known data pattern, or pilot, is used in the search correlations to increase the length and accuracy of the search. Because this reference pattern is the same for all radio paths between the mobile and the base station, it should be calculated only once before it is used to correlate.

2.2 Rake Energy Correlations

Only the pilot bits of the control channel are known. These are transmitted on the Q channel, but the received signal is of unknown phase relative to the transmitted signal; therefore, cross-correlation must be performed on both the I and Q channel (see Figure 1 and equations (1) and (2)).

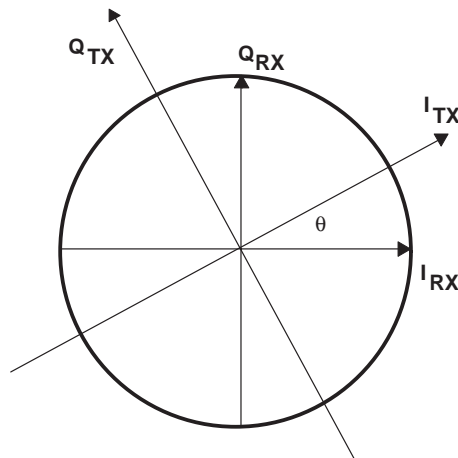


Figure 1. Phase Rotation

$$I_{RX} = I_{TX} \times r \times \cos \theta - Q_{TX} \times r \times \sin \theta \quad (1)$$

$$Q_{RX} = Q_{TX} \times r \times \cos \theta + I_{TX} \times r \times \sin \theta \quad (2)$$

Because the transmitted I channel is known to be orthogonal to the transmitted Q channel and correlation is performed on the Q channel reference, the Q channel correlates to ± 1 and the I channel correlates to 0. Therefore, the previous equation becomes:

$$I_{RX} = -r \times \sin \theta \quad (3)$$

$$Q_{RX} = r \times \cos \theta \quad (4)$$

For any θ , $\cos^2\theta + \sin^2\theta = 1$; therefore, by squaring and adding cos and sin, you can obtain r^2 .

Because the purpose of all these correlations is to find the maximum paths, when you compare the squares, you are also comparing the real values; therefore, there is no need to compute the square root to obtain the real r .

If the path is valid, the values in I_{RX} and Q_{RX} can be used to calculate weights for the I and Q channels when they are combined for data extraction later. Because the individual channels are weighted according to their importance, only a crude normalization is done initially. Small errors are corrected by the convergence of the weighting factors.

2.3 Path Selection

A single correlation run is likely to pick up both valid and invalid paths; therefore, it is important to use a good search strategy. There are two possible scenarios:

1. Valid path information is known from previous frames. (tracking)
2. Delay information is known, but no valid or invalid path information is known from previous frames. (training)

2.3.1 Tracking

To track the valid paths, it is important to use the known information. Look at each path individually and compare it with the correlation obtained for the previous oversampling period and the next oversampling period.

In the ETSI-UMTS specification, the maximum speed is 800 Km/h (500 mph), which corresponds to a movement of 2.2 m/frame, where a frame is 10 ms. The oversampling rate is 32 MHz (8 x 4 MHz), and the equivalent distance between oversampling points is 9.15 m. The minimum time that a mobile station can move from one oversampling point to the next is 4 frames. Tracking is tested for the pilot data on the first slot of every frame. If, for over 4 frames, the previous or next slot is better, the path is moved or tracked to that location. This allows paths to be tracked and prevents any noise that may produce an erroneous track.

New paths may become stronger than the existing paths; for this reason, a search of a subsection of the possible delays is used to find a new path. Radio propagation theory shows that new paths are likely to be near the strongest paths or after them (see Figure 2).

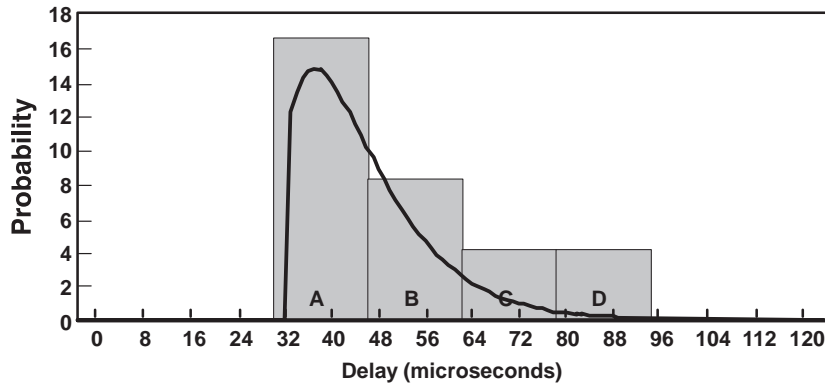


Figure 2. Path Search Sequence

The search pattern is therefore biased to search in these areas. Due to movement, it is important to search the same area on subsequent frames so that paths do not move before that area is searched again. The search sequence is AABBAACC-AABBAADD. (That is, each area is searched twice in succession to allow for correlation between successive frames, area A is searched twice as often as area B, and area B is searched twice as often as either area C or D, thereby biasing the search to shorter paths.)

The area before A is ignored because the mobile station is too far away to have a reflection in this area. The area after D is ignored because the path is too long to have a significant signal, due to the long path distance and the high angle of reflection. When the new path is found, it replaces the poorest old path. The new path is given an initial weighting, equivalent to the second poorest old path because this allows the new path a chance to establish itself as better than this before the next update. Overall, this search technique is eight times more efficient than a full search and loses only 1–2% of unusual paths.

2.3.2 Training

In the second case, either no path information is known or the paths changed completely due to a change in obstacles between the mobile and base station. However, the rough distance to the mobile station is known from either the random access channel time or the previous minimum delay path. The old minimum delay path gives a rough estimate of the mobile station position. Assuming the mobile station has not moved far, a complete search from immediately prior to this delay is performed, and a complete set of new paths is generated. Figure 3 shows the search area and the probability function.

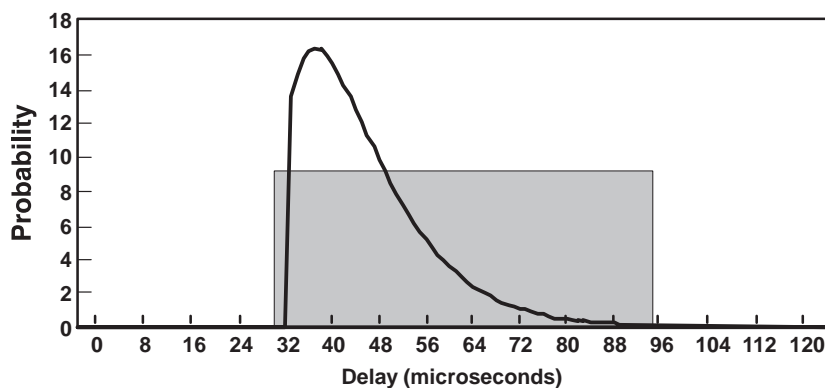


Figure 3. Path Probability Tracking

In the ETSI system model, the active delay period is 31 μ S in a 125- μ S cell radius.

Using the pilot, a number of paths greater than those actually required are initially selected, and all delays at 1/2 the chip period are searched. When these rough matches have been found, further correlations are done at the oversampling rate (1/8 of the chip period) near to the paths to find the eye maximum. To increase the energy for valid paths, correlations are then performed that include the remaining control and data bits. The correlation results are squared to remove the transmitted data, and the final paths are then selected. Because no performance information is available, these paths are allocated initial weights based on their received energies. With this strategy, the MIPS are reduced by 98.25% to 1.75% (25% * 25% * 30%).

- 75% reduction for 1/2 chip period versus all oversamples (oversample = 8)
- 75% reduction for the path estimate
- 70% reduction for using the 6-bit pilot signal for the preliminary search (95% if only a 3-bit pilot portion is used)

Some of this reduction is later wasted by the need to repeat some of the incomplete correlations so the overall performance is 97–98% better than the brute force full search approach. In low channel utilization cells (<50%) where only a few mobile stations are likely to be operating, the amount of the pilot signal used can also be reduced from 6 bits to 3 bits, which further reduces the MIPS load.

In both cases, to extend the correlation time, the correlated energy over several frames is averaged with a 1, 1/2, 1/4, ... sequence. This effectively doubles the correlation time and reduces false positives while still allowing for mobile station movement. After the path is found, the search energy returns to separately correlated I and Q channels. Based on these energies and the reference pilot data, the phase rotation parameters are calculated for decoding.

2.4 Rake Decoding

A data bit correlation is performed for each of the data bits in each of the known data paths. For each bit, the paths are weighted with the old weights and combined. The result for each path is compared with the weighted result. If they are the same, the weight for that path is increased by an amount proportional to the energy in that bit. If they are different, however, the weight for that path is decreased. In this way, path weights converge based on strength and accuracy. Gradually, this replaces the original estimate based only on strength. The energy in the I and Q components is also filtered relative to the expected I and Q phases so that slow phase changes caused by movements are tracked.

2.5 Radio Feedback

In addition to performing the data demodulation, the rake receiver provides a feedback parameter of the total energy in all the fingers, and this parameter can be used to control the gain plan for this and other channels.

3 Implementation

To reduce the emulation time and allow long simulations, the implementation of a rake receiver uses slot-by-slot processing with divided tasks for the host PC and the TMS320C62x™ DSP evaluation module (EVM).

On the PC side, the ManyMobileCDMAs.exe program takes full control of the EVM. The data transfer for the generated test I/Q data, the output of the rake receiver, as well as associated control flags, and the initialization values for the rake receiver are all handled via the host port interface (HPI) of the DSP. Additional DSP memory buffers are needed to ensure the continuous execution of the rake receiver on a slot-by-slot basis. The control input data (for example, for reference and output file names) searches channel numbers and is sent by the ManyMobileCDMAs.exe file originating from a text control file. This can include control data for several different rake receiver processes with different parameters.

Figure 4 shows the flowchart diagram for the execution of the ManyMobileCDMAs.exe program and its communication with the TMS320C62x™ DSP EVM.

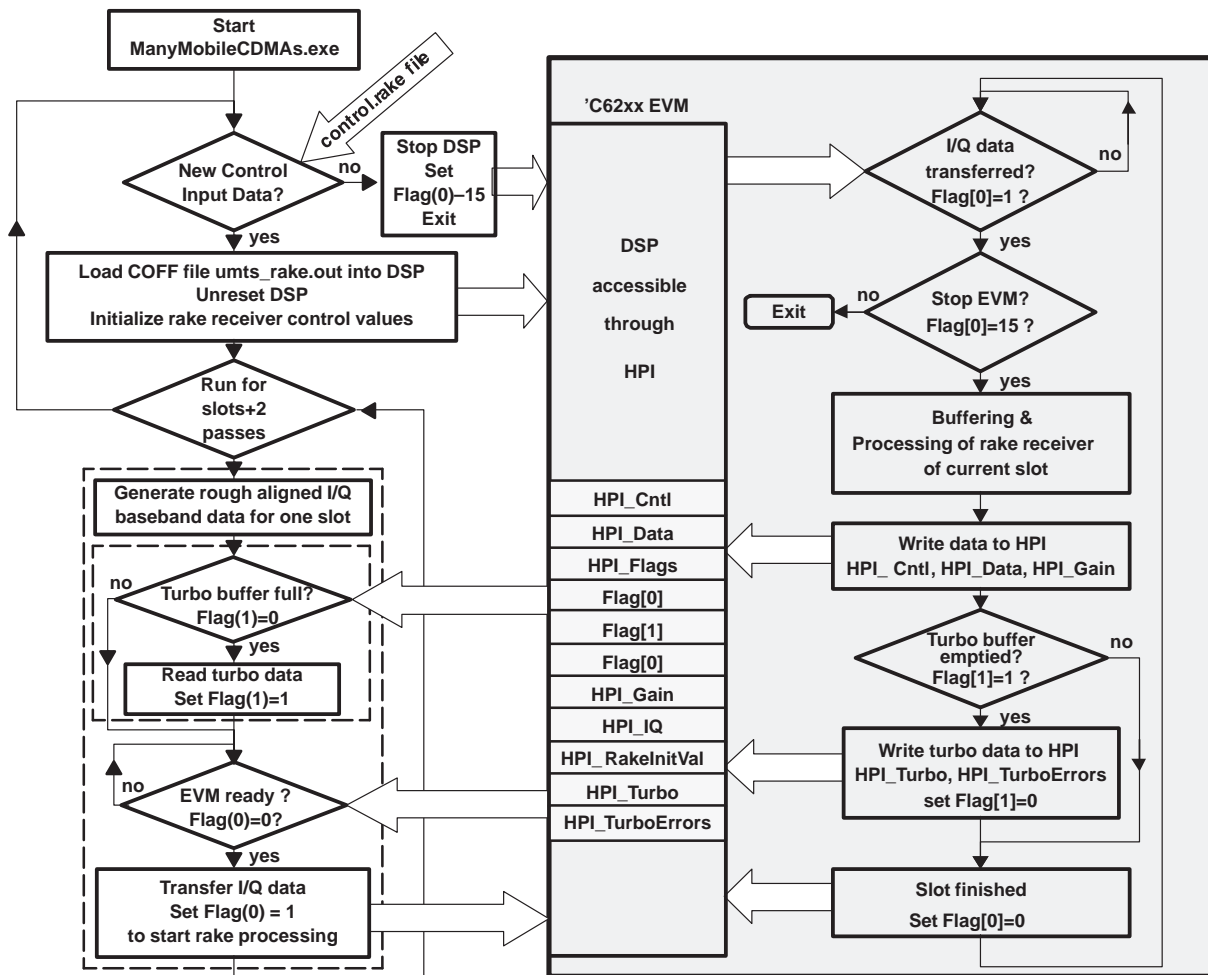


Figure 4. Flowchart of Rake Receiver Simulation and Communication Between the PC and the TMS320C62x™ DSP EVM

3.1 Selection of Rake Receiver Simulation With the TMS320C62x™ DSP EVM

The control file ManyMobileCDMAs.exe file can be configured to generate output data for the TMS320C62x™ simulator using CIO or it can directly control the TMS320C62x™ DSP EVM. For EVM mode, the OUT_HPI option must be assigned to OUT_MODE in both the control.cpp file (compiled and linked to ManyMobileCDMAs.exe) and in the control.c file (compiled and linked to umts_rake.out).

If OUT_HPI is chosen and the associated files are compiled, slot-by-slot processing with divided tasks for the host PC and the TMS320C6x™ DSP EVM can be performed. The ManyMobileCDMAs.exe file takes full control of the EVM.

3.2 TMS320C62x™ DSP EVM Memory Mapping

Due to the HPI usage and the introduction of additional buffers, please be aware of the considerations listed in Table 2 regarding memory mapping.

Table 2. TMS320C62x™ EVM—PC Communications With HPI Common Memory Map: Data Changes for Each Slot

Control.c (C6x CONTROL FILE)HPI VARIABLES	EVM→PC PC←EVM	ManyMobileCDMAs.exe HPI HEX MEMORY ADDRESS/ CORRESPONDING VARIABLE(s)	SIZE IN BYTES (HEX/DEC)	DESCRIPTION
HPI_Cntl	→	02000008/CntlRead[0...639]	280/640	Output array of received control
HPI_Data	→	02000288/DataRead[0...639]	280/640	Output array of received data
HPI_Flag[0...7] HPI_Flag[0] HPI_Flag[1]	↔	02000000/Flag[0...7] 02000000/Flag[0] 02000000/Flag[1]	8/8 1/1 1/1	Only 2 flags used so far [0] ready bit for HPI r/w [1] ready bit turbo data buffer full/empty
HPI_Gain	→	020145dc/Gain	8/8	Output gain control
HPI_IQ	←	02000508/IQ_Frame[0...256*10*8]	14000/ 81920	Input array of rough aligned baseband data
HPI_RakelnitVal	←	020145c8/RakelnitVal[0...3] RakelnitVal[0]=Data Channel No RakelnitVal[1]=Control Channel No RakelnitVal[2]=Data Spread RakelnitVal[3]=Control Spread	14/20	Input array of rake receiver init values
HPI_Turbo	→	02014508/Turbo[0...191]	C0/192	Output array of turbo decoder data
HPI_TurboErrors	→	020145e0/TurboErrors	8/8	Output turbo decoder errors

Communication through these buffers is controlled via the two control files called Control.cpp in the ManyMobileCDMAs.exe file and control.c in the umts_rake.out file.

Slot-by-slot processing of the rake receiver requires a buffer on the TMS320C62x™ DSP EVM side:

```
int buffer_x[2560*Ne+Max_Test];
```

Because this buffer array is located in the .far data section, you must ensure that you have enough memory space by using the linker command.

3.3 Buffering Scheme

The ManyMobileCDMAs.exe program generates 2560*Ne bits for each slot. These bits are transferred to the HPI to be accessible by the rake receiver software running on the TMS320C62x™ DSP EVM. Because the rake search requires more bits than the current slot, a buffering scheme is applied.

Figure 5 demonstrates why two more slots must be processed in order to read the last result of the rake receiver output. Rake receiver execution starts one slot late to allow access to the previously generated I/Q data and the currently generated I/Q data.

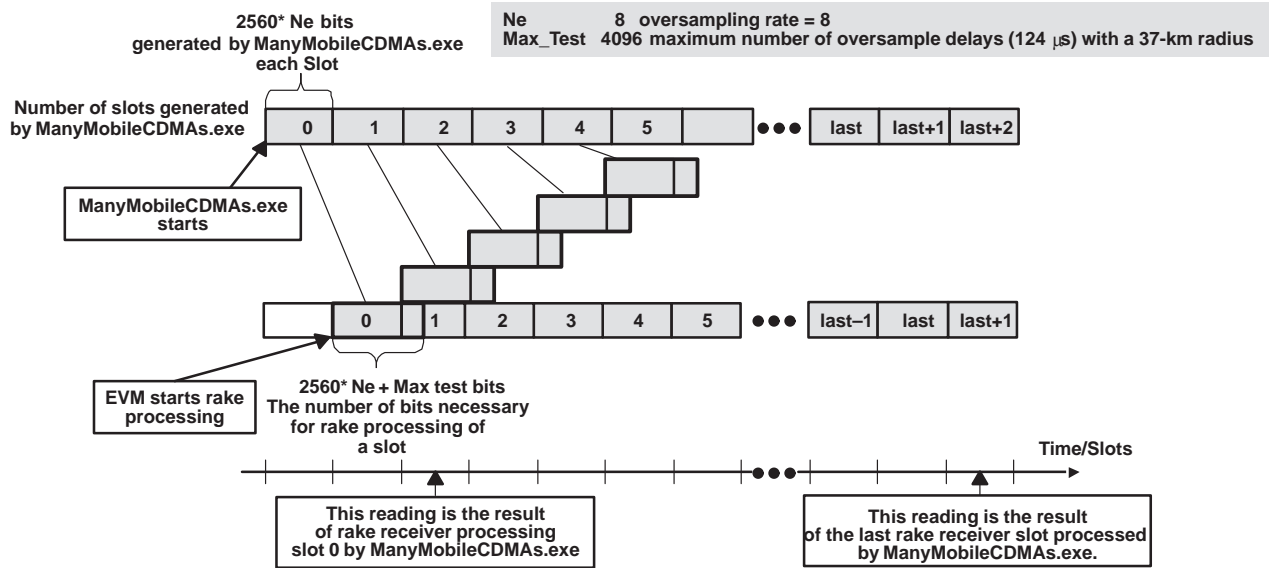


Figure 5. Buffering of Rake Receiver Input Data

3.4 TMS320C62x™ DSP EVM Usage

The host port interface (HPI) access from the PC by the ManyMobileCDMAs.exe program uses the TMS320C62x™ DSP EVM host support software that provides low-level Windows™ drivers in a Win32 DLL (evm6x.dll, evm6xdll.h) environment.

The exact use can be found in Chapter 2 of the *TMS320C6201/6701 Evaluation Module Technical Reference Guide*. This chapter shows how to write and read data to and from the HPI as well as how to load the COFF files and run the code.

Link cvectors.asm to your .out file to ensure the right behavior during the start-up of the DSP. All EVM DIP switches are set to the factory default, but only the JTAGSEL is set to Internal. Memory map 1 is applied.

3.5 Control File Control.Rake Structure

The control.rake file contains all input data (10 lines) for an emulation. The ManyMobileCDMA.exe file continues to read this file as long as there is control input data available; that is, several different emulation environments can be executed in a row. Every line of data requires a carriage return at the end of the line. Example 1 describes the structure of the control rake file.

Example 1. Control.Rake File Structure

- Line 1
 - –CSV file, radio output data reference file. (Write.)
 - –Format binary 1,-1

- Line 2(OUT_MODE=OUT_HPI)
 - CSV file, Radio received bits from data and control channel. (Write.)
 - Format signed 8 bit weights , +ve=1, -ve=0,
 - Absolute value = weight, 127 good likelihood 1, 1 bad likelihood 1
 - Absolute value = weight, -127 good likelihood 0, -1 bad likelihood 0
- Line 2(OUT_MODE=OUT_FILE)
 - BIN file, binary radio output data WCDMA encoded data. (Write.)
 - Format oversampled 32-bit integers with 16-bit signed I channel in high half and 16-bit signed Q channel in low half.
 - Length n slots (5120*8*n) 5120 = chip rate, 8 = oversample n = 1,2,3, ...
- Line 3
 - CSV file, turbo encoded reference data channel. (Write.)
 - Format binary 1,0.
 - (Ignored by the TMS320C62x™ when simulating)
- Line 4
 - CSV file, turbo decoded received data channel. (Write.)
 - Format binary 1,0.
 - Ignored if OUT_MODE=OUT_FILE
- Line 5
 - Number of channels
 - Maximum 128
 - (Ignored by the TMS320C62x™ when simulating)
- Line 6
 - Data channel spreading factor
 - 4,8,16,32,64,128,256
- Line 7
 - Control channel spreading factor
 - 4,8,16,32,64,128,256
- Line 8
 - Data channel channelization code
 - 0–255

- Line 9
 - Control channel channelization code
 - 0–255
- Line 10
 - A/D Accuracy mask
 - FF00FF00 (8-bit)
 - FC00FC00 (6-bit)
 - F000F000 (4-bit)
- Repeat Lines 1–10 for multiple runs.

3.6 Control.cpp (Generate on PC)

- Inputs from PC

The control.cpp file reads the control.rake file to determine rake parameters and filenames for a particular run. This file performs all file input and output operations and the communication with the DSP via the HPI. (See the control.rake file in Example 1 for more details.)

- Return code to PC
 - 0 Execution completed.
 - Bit 0 = 1 Invalid data channel channelization code.
 - Bit 1 = 1 Invalid control channel channelization code.
 - Bit 2 = 1 Invalid data channel spreading function.
 - Bit 3 = 1 Invalid control channel spreading function.
 - Bit 4 = 1 Control.rake file not found.
 - Bit 5 = 1 Input reference file, write file error.
 - Bit 6 = 1 Output file, radio received bits from data and control channel, write file error.
 - Bit 7 = 1 Turbo decoded reference data channel, write file error.
 - Bit 8 = 1 Turbo decoded received data channel, write file error.
 - Bit 9 = 1 Unable to open EVM board
 - Bit 10 = 1 Unable to reset EVM board or DSP
 - Bit 11 = 1 Unable to open the HPI port on the TMS320C62x™ DSP EVM
 - Bit 12 = 1 Unable to init EMIF
 - Bit 13 = 1 Unable to load COFF file
 - Bit 14 = 1 Unable to unreset TMS320C6x™ DSP EVM
 - Bit 15 = 1 Not used
- Multiple errors may set multiple bits.

- Outputs to PC

Raw received radio data for control channel as weighted bytes

Raw received radio data for data channel as weighted bytes

Turbo-decoded binary data for data channels as 0,1 bytes

Measured gain plan performance. (Optimum receive is 32768.) This return value should be used to perform gain control feedback both for transmitted power and receive AGC. If this value is high for multiple channels, then the receive AGC is too high. If this value is high for only one channel, then the transmit power of that channel is too high. If this value is low for only one channel, then the transmit power of that channel is too low.

Turbo errors are corrected in turbo frame. This return value should be used to judge when a retrain/hand-over is required due to a discontinuity in reception.

3.7 Control.c (Receive on TMS320C62x™ DSP)

This program provides the control functions from the TMS320C62x™ DSP EVM side of the link, and it also initializes the EVM.

- Inputs

HPI_RakelnitVal: Initialization data to tell receiver channel and spreading information

HPI_IQ: Input radio binary data

- I/Os

HPI_Flag: Flags to control transfer with control.cpp

- Outputs

HPI_Cntl: Received control channel information

HPI_Data: Received data channel information

HPI_Gain: Received signal strength level

HPI_Turbo: Received turbo-decoded data from data channel

HPI_TurboErrors: Number of errors that the turbo decoder has corrected.

3.8 Rake.c

The rake.c file contains all the subroutines and constants used by the rake radio receiver.

The const variables are channel independent and may be reused.

The .bss variables are channel-dependent and must be preserved in a multichannel environment.

3.8.1 *Rake_Reset (global)*

The rake_reset subroutine initializes the rake receiver algorithm and should be called when a call is set up, handed over, or when a retrain is requested from the high-level code.

- Function

The external function is defined: – void Rake_Reset();

3.8.2 *Rake_Energy (local)*

This subroutine measures the energy in a potential bit or pilot bit sequence without affecting the long-term path estimations. This subroutine is used for fine searching and data decode.

The return value is given by:

$$\left(\sum_{i=0}^{SF-1} Code_{Channel}[i] \times X_I[i \times Ne] \right)^2 \quad (5)$$

Where:

X is a starting address in the complex I/Q received radio data.

Code is the reference word being searched for.

SF is the number of reference chips over which correlation is being performed.

3.8.3 *Rake_Energy_Update (local)*

This subroutine measures the energy in a potential bit or pilot bit sequence and updates the long-term path estimations so as to average the energy over several frames. It is used mainly for course searching.

The return value is given by:

$$New_I = \sum_{i=0}^{SF-1} Code_{Channel}[i] \times X_I[i \times Ne] \quad (6)$$

$$New_Q = \sum_{i=0}^{SF-1} Code_{Channel}[i] \times X_Q[i \times Ne] \quad (7)$$

$$I[k] = I[k]/2 + New_I \quad (8)$$

$$Q[k] = Q[k]/2 + New_Q \quad (9)$$

$$I[k]^2 + Q[k]^2 \quad (10)$$

Where:

X is a starting address in the complex I/Q received radio data.

Code is the reference word being searched for.

SF is the number of reference chips over which correlation is being performed.

k is a reference distance associated with X.

3.8.4 *Rake_Search (global)*

This subroutine checks all known paths for bit slippage due to paths moving. It searches for a new path using current and historic data in part of the delay profile and replaces the poorest performing old path with the strongest new path.

The initial weight for the new path is that of the second poorest old path.

- Function

The external function is defined as follows:

```
void Rake_Search (const int* x, const char* c_DPCCH, const short SF_DPCCH, const short PilotLen);
```

x: Input array of base1-band data, length 2560*Ne, Q channel in high half; I channel in low half

c_DPCCH: Input array of reference code word for control, length 2560 (scrambled, channel code)

SF_DPCCH: Spreading function for control, {4,8,16,32,64,128,256}

PilotLen: Pilot length for search in bits

3.8.5 *Rake_Srch_1 (global)*

This subroutine searches for paths in all data delays and replaces all paths with new paths. Based on information from the delay path of either the random access channel (RACCH) or the last path knowledge before a discontinuity in paths, a 31- μ S window within the 124- μ S cell radius is searched at an oversampling rate of twice the chip rate. Only the pilot data is searched for. From all these paths, the 20 strongest signals are selected as candidate paths. For each of these candidates, the search pattern is fine tuned to the 8 times oversampling rate, and the energy from the remaining bits in both the data and control channels is added to these estimates. The final 6 paths are then chosen from the original 20 candidate paths, and each path is assigned an initial weight based on the square energy in the channel.

For the first few frames after a reset, hand-over, or retrain-request, this call is used instead of Rake_Search. All past data on paths is ignored.

- Function

The external function is defined as follows:

```
void Rake_Srch_1 (const int* x, const char* c_DPDCH, const char* c_DPCCH, const short SF_DPDCH, const short SF_DPCCH, const short PilotLen);
```

x: Input array of baseband data, length 2560*Ne, Q channel in high half, I channel in low half

c_DPDCH: Input array of reference code word for data channel, length 2560 (scrambled, channel code)

c_DPCCH: Input array of reference code word for control, length 2560 (scrambled, channel code)

SF_DPDCH: Spreading function for data, {4,8,16,32,64,128,256}

SF_DPCCH: Spreading function for control, {4,8,16,32,64,128,256}

PilotLen: Pilot length for search in bits

3.8.6 *Rake_Receive (global)*

For each known rake receiver, *Path*, and for each transmitted bit *n*:

Extract data from all known paths:

$$I_{Data}[Path, n] = \sum_{i=0}^{SF_{Data}-1} Code_{DataChannel}[i + n \times SF_{Data}] \times X_I[T[Path] + i \times Ne + n \times SF_{Data} \times Ne] \quad (11)$$

$$jQ_{Data}[Path, n] = \sum_{i=0}^{SF_{Data}-1} Code_{DataChannel}[i + n \times SF_{Data}] \times X_{jQ}[T[Path] + i \times Ne + n \times SF_{Data} \times Ne] \quad (12)$$

Normalize data channel:

$$I_{Data}[Path, n] + jQ_{Data}[Path, n] = norm(I_{Data}[Path, n] + jQ_{Data}[Path, n]) \quad (13)$$

Rotate data channel:

$$PathBit_{Data}[Path, n] = I_{Data}[Path, n] \times \sin(-\vartheta[Path]) - jQ_{Data}[Path, n] \times j\cos(-\vartheta[Path]) \quad (14)$$

Weight Data Channel Rake Paths

$$DataBit[n] = \sum_{Path=1}^{Nb_fingers} PathBit_{Data}[Path] \times Weight[Path] \quad (15)$$

Extract control from all known paths:

$$I_{Control}[Path, n] = \sum_{i=0}^{SF_{Control}-1} Code_{ControlChannel}[i + n \times SF_{Control}] \times X_I[Path + i \times Ne + n \times SF_{Control} \times Ne] \quad (16)$$

$$jQ_{Control}[Path, n] = \sum_{i=0}^{SF_{Control}-1} Code_{ControlChannel}[i + n \times SF_{Control}] \times X_{jQ}[Path + i \times Ne + n \times SF_{Control} \times Ne] \quad (17)$$

Normalize control channel:

$$I_{Control}[Path, n] + jQ_{Control}[Path, n] = norm(I_{Control}[Path, n] + jQ_{Control}[Path, n]) \quad (18)$$

Rotate control channel:

$$PathBit_{Control}[Path, n] = I_{Control}[Path, n] \times \cos(\vartheta[Path]) - jQ_{Control}[Path, n] \times j\sin(\vartheta[Path]) \quad (19)$$

Weight control channel rake paths:

$$ControlBit[n] = \sum_{Path=1}^{Nb_fingers} PathBit_{Control}[Path] \times Weight[Path] \quad (20)$$

Where:

X is the start address of a slot of received data as I + jQ.

T [Path] is an array of path delays for the rake fingers.

Ne is the oversampling rate.

Code is the reference word being searched for (data and control channels).

SF is the number of reference chips over which correlation is being performed for each received bit (data and control channels). These reference chips are premixed from the orthogonal channel and the scrambling code before correlation.

θ [Path] is the phase difference between the radio reference and the received signal.

Weight [Path] is the weight applied to data on that path.

Updates the weights based on whether the path produced the same result as the vote

Updates the phase adjustment to allow for movement of the mobile station

- Function

The external function is defined as follows:

```
int Rake_Receive (const int* x, const char* c_DPDCH, const char* c_DPCCH, const short
SF_DPDCH, const short SF_DPCCH, char* y_DPDCH, char* y_DPCCH, int fingers);
```

x: Input array of baseband data, length 2560*Ne, Q channel in high half, I channel in low half

c_DPDCH: Input array of reference code word for data channel, length 2560 (scrambled, channel code)

c_DPCCH: Input array of reference code word for control, length 2560 (scrambled, channel code)

SF_DPDCH: Spreading function for data, {4,8,16,32,64,128,256}

SF_DPCCH: Spreading function for control, {4,8,16,32,64,128,256}

y_DPDCH: Output array for data, length 2560/SF_DPDCH {640,320,160,80,40,20,10}

y_DPCCH: Output array for control, length 2560/SF_DPCCH {640,320,160,80,40,20,10}

Fingers: Number of fingers to decode

3.9 Scramble.c

This file contains all subroutines and constants used by the orthogonal channelization and scrambling functions. The const variables are channel independent and may be reused.

The const unsigned int ChanRom[256][8] defines the 256 different channelization codes as bit codes. Bits are coded into words LSB (bit0) first. If they are read as shorts or bytes, little endian mode must be used.

The const unsigned int ChanTable[16] is used to expand the binary 1/0 of the mixed scrambler and channelization codes into 1/-1 format needed for correlation in the rake receiver. These are written as words and read as bytes.

The .bss variables are channel dependent and must be preserved in a multichannel environment.

3.9.1 *Scramble_Init (global)*

This file initializes the scrambling algorithm to a known point and initializes the circular buffer. It also calls for slot 0 to reinitialize the scrambling for each frame. This known point corresponds to a 40-bit scrambling code number.

- Function

The external function is defined as follows:

```
void Scramble_Init (unsigned int n_high, unsigned int n_low);
```

Where:

n_high: Input bits for scrambling code number; bits 0–8 correspond to bits 32–40 of scrambling code number

n_low: Input bits for scrambling code number; bits correspond to bits 0–31 of scrambling code number

3.9.2 *Scramble (global)*

This function generates reference signals for the rake receiver based on the channel number and the scrambling code. One reference signal is produced for the data channel and two signals are produced for the control channel. One of the control channel codes is true data for decoding the control channel, whereas the second control channel has the pilot bits encoded to simplify correlation over the pilot bits when a new path is being searched for in the rake receiver. This function must be called for each slot.

- Function

The external function is defined as follows:

```
void Scramble(const int Pilot, const int CntlChannel, const int DataChannel, const short SF_DPDCH, const short SF_DPCCH, char* c_DPDCH, char* c_DPCCH, char* c_Pilot)
```

Where:

Pilot: Expected pilot bit code

CntlChannel: Control channel spreading code

DataChannel: Data channel spreading code

SF_DPDCH: Spreading function for data channel {4,8,16,32,64,128,256}

SF_DPCCH: Spreading function for control channel {4,8,16,32,64,128,256}

c_DPDCH: Output array of reference code word for data channel length 2560 (scrambled, channel code)

c_DPCCH: Output array of reference code word for control channel length 2560 (scrambled, channel code)

c_Pilot: Output array of reference code word for control channel (Pilot) length 2560 (scrambled, channel code)

3.9.3 Implementation of Scrambling Code Generation

The scramble function generates 32 new bits each loop pass.

- Initialization of scrambling code generator (Scramble_Init):

The given initialization values (n_{high} , n_{low}) are used and additional bits are produced in order to provide 32-bit packets of each following Scramble call.

The buffer Old_a is filled with thirty-two 32-bit values, that is, 1024 bits to generate data and control reference code words (I and Q components) in a cyclic consecutive way later on.

- Scrambling code generation:

The scrambling code sequences are constructed as the position wise modulo 2 sum (XOR) of two binary m-sequences, the x and y sequences, and each one generated by polynomials of degree 41.

- x-sequence

Polynomial: $1+X^3+X^{41}$

The binary recursive definition results in the following equation:

$$x(41 + n) = x(0 + n) \wedge x(3 + n) \quad (21)$$

Where:

$x(\dots)$ represents a bit of the x-sequence.

Because the distance between the bits, $x(3+n)$ and $x(41+n)$, is greater than 32, a simple implementation with 32-bit integer values is possible. Previously produced x-sequence bits must be arranged to result in a modulo-2 sum in the next 32 x-sequence bits to ensure a cyclic simultaneous generation.

Figure 6 shows the initialization phase that assigns the 40-bit scrambling code number ($n_{40}-n_0$) to bit 0 to 40 and generates the missing bits to start with the cyclic execution of scramble at bit 64.

After that, each loop pass takes previous x-sequence bit fields and puts the two components together to get the next 32 x-sequence bits. These bits are always aligned to 32-bit boundaries.

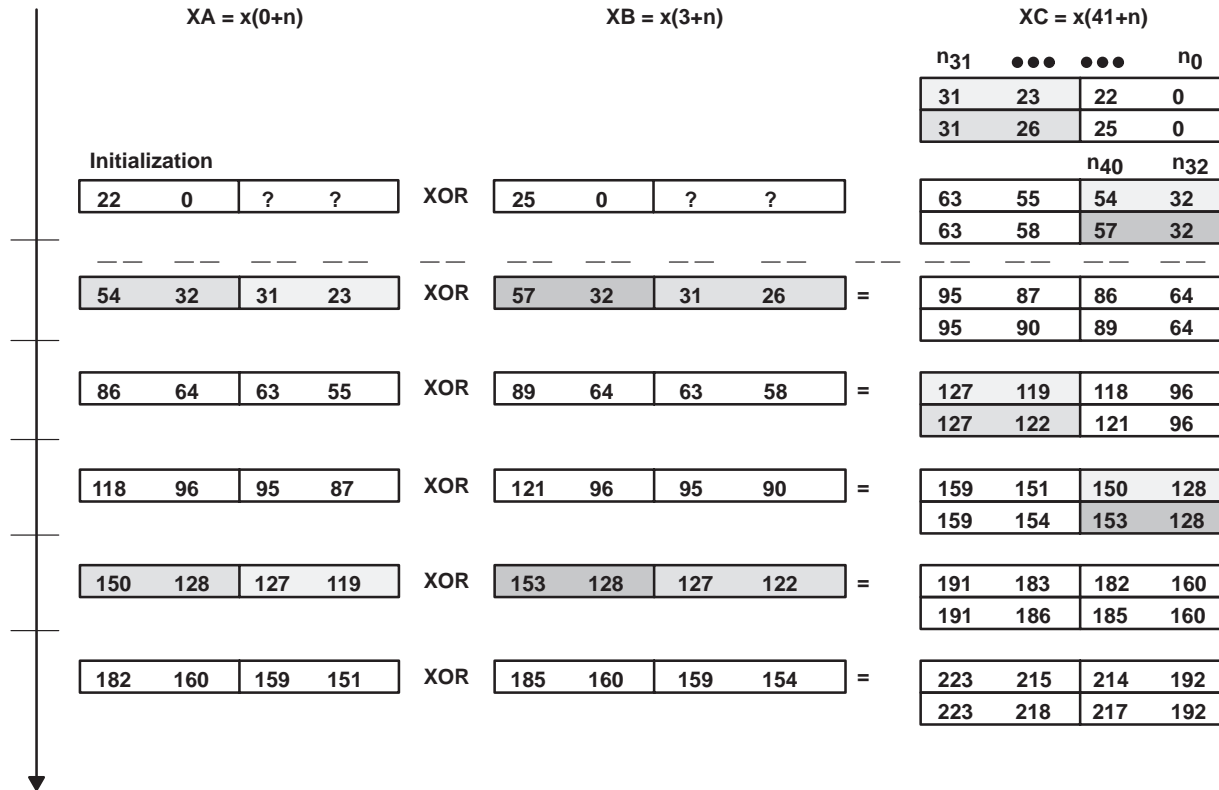


Figure 6. Initialization and Beginning of Consecutive x-Sequence Generation

- y-sequence

Polynomial: $1+X^{20}+X^{41}$

The binary recursive definition results in the following equation:

$$y(41 + n) = y(0 + n) \wedge y(20 + n) \tag{22}$$

Where:

y(...) represents a bit of the x-sequence.

In this case, the generation of new bits takes more cycles compared with the x-sequence because there are not enough previous y-sequence bits from former loop passes available. As shown in Figure 7, the XOR operation must be performed twice, first to generate the most significant bits (MSB) of YB (second component of modulo 2 sum), and second to get new 32 y-sequence bits.

The provision of all the other needed bit fields works similar to the x-sequence generation.

The initialization assigns 1 to every bit from bit number 0 to 40.

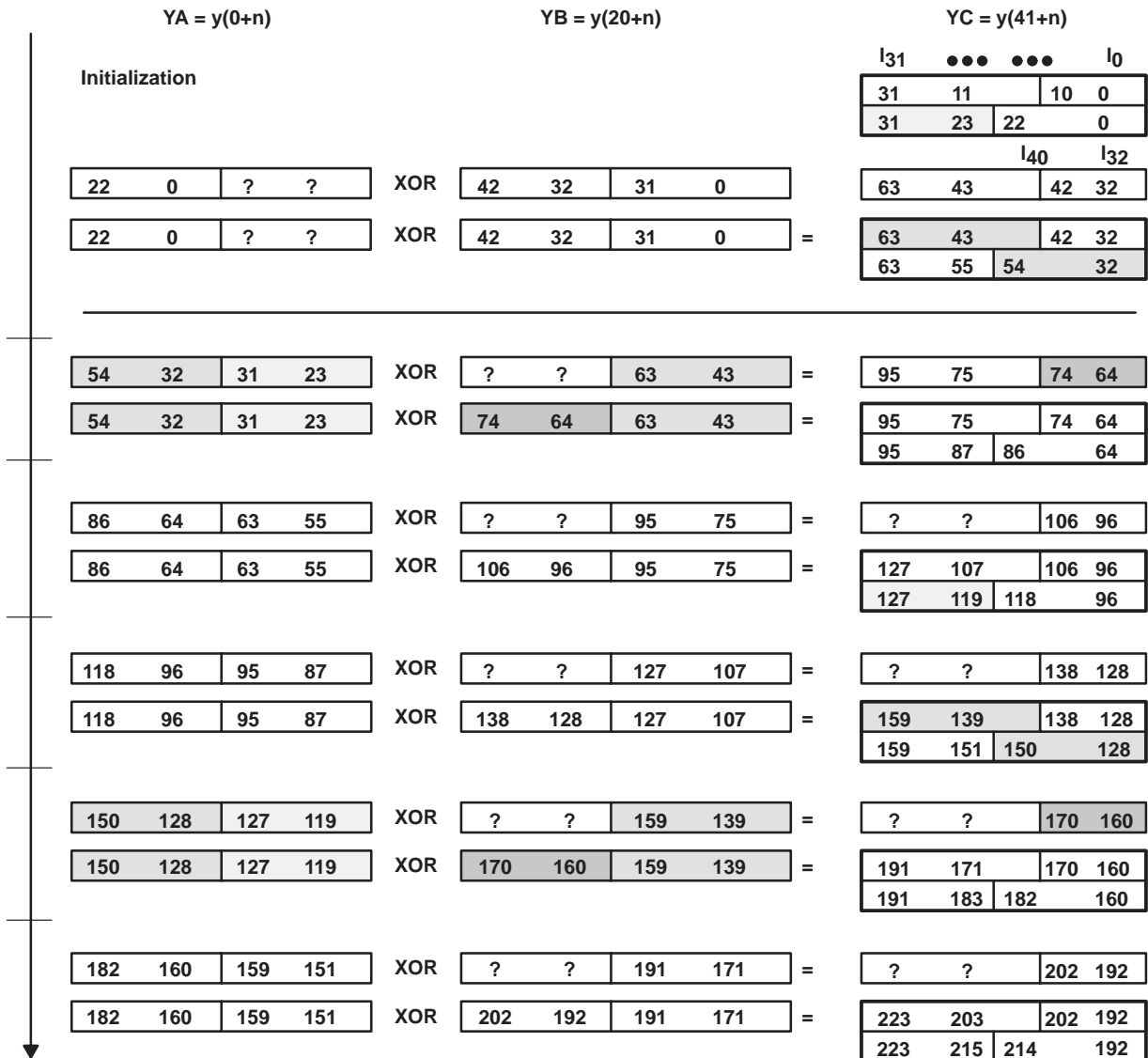


Figure 7. Initialization and Beginning of Consecutive y-Sequence Generation

- Scrambling code word

The scrambling code word is a combination of the x- and y-sequence, created by adding the corresponding bits as a modulo-2 sum.

The binary recursive definition results in the following equation:

$$a(41 + n) = x(41 + n) \wedge y(41 + n) \tag{23}$$

or

$$a(64 + n) = x(64 + n) \wedge y(64 + n) \tag{24}$$

Where:

$a(\dots)$ represents a bit of the scrambling code word.

- Scrambling code word generation for in-phase (I) and quadrature (Q) components

In-phase and quadrature components use the scrambling code word bits, but the quadrature components are a 1024-bit shifted version of the in-phase scrambling code. The 1024 bits must be buffered (array `Old_a`) to enable the cyclic simultaneous generation of I and Q scrambling code words.

- Channelization

The scramble function not only generates the scrambling code word, this code word already includes the channelization data.

The channelization codes are orthogonal variable spreading factor (OVSF) codes. A code tree can generate these codes.

The pre-generated channelization data is stored in the `ChanRom` table as 0/1 values. Therefore, only an additional XOR operation with the scrambling code words can create the reference code word.

- Conversion 0/1 \rightarrow $-1/1$

The reference code word bits must be available as $-1/1$ values in a character array. Therefore, each bit of this reference code word is converted to a $-1/1$ byte by using the `ChanTable` table that consists of sixteen 32-bit values. Each byte of the 32-bit value holds the conversion value for one bit of the reference code word.

Each conversion generates four characters ($-1/1$) by extracting four bits of the I or Q reference code word. These four bits select an element of the `ChanTable` table (consisting of 32-bit values \rightarrow 4 byte). This corresponding element is stored to the output array, which is later used as character array.

- Control channel with pilot

To simplify correlation over the pilot bits, a second control output array (including the pilot bits) is generated by use of the given pilot bit code.

3.10 `Turbodec.c`

This file contains all the subroutines and constants used by the turbo decoding algorithm. This code is fully re-entrant because there are no channel-specific variables. The weighted bytes are turbo decoded with a max-log-map Viterbi decoder to produce the original data.

- Function

The external function is defined as follows:

```
void turbo_decoder(const channel_t* y_sys, const channel_t* y_par1, const channel_t*
y_par2, Binary* hd);
```

`y_sys`, `y_par1`, `y_par2`: 3 arrays of 195 bytes of turbo encoded data. Each byte contains a weighted bit $+127 \dots -127$. The more positive the weighted bit, the stronger the one received from the radio; the more negative the weighted bit, the stronger the zero.

`hd`: 192 bytes of decoded data with each byte containing 1 bit as hard binary data 0/1.

3.11 Turboenc.c

This file contains all subroutines and constants used by the turbo encoding algorithm. After turbo decoding, the data is re-encoded to measure the number of errors corrected by the turbo decoding. This parameter can then be passed to the system control software to determine if hand-over or gain adjustment is required. This code is fully re-entrant because there are no channel-specific variables.

- Function

The external function is defined as follows:

```
void turbo_encoder (Binary* sys, Binary* p1, Binary* p2);
```

sys: Input – 192 bytes of data with each byte containing one bit as hard binary 0/1

Output – 195 bytes of data with each byte containing one bit as hard binary 0/1.

p1/p2: 2 arrays of 195 bytes of turbo encoded data. Each byte contains one bit as hard binary 0/1.

3.12 Turbo.h

This header file defines the system configuration for the turbo encoder and decoder.

3.13 Umts.h

This header file defines all the global subroutines and system level options for different rake receivers.

4 Algorithm Performance

The performance of the code was verified by simulating the radio signals produced by many mobile stations operating within a single cell on a single CDMA frequency band. Mobile stations were positioned in this cell using the Monte Carlo modeling method in which each mobile was allocated 6 paths with random time delays and phases:

- 0dB±0.4dB Delay 0 ... 255 oversample range
- -1.2dB±0.4dB Delay 0 ... 511 oversample range
- -2.5dB±0.5dB Delay 0 ... 767 oversample range
- -4.1dB±0.6dB Delay 0 ... 1023 oversample range
- -6.0dB±0.8dB Delay 0 ... 1279 oversample range
- -8.5dB±1.0dB Delay 0 ... 1535 oversample range

The path descriptions are also illustrated in Figure 8.

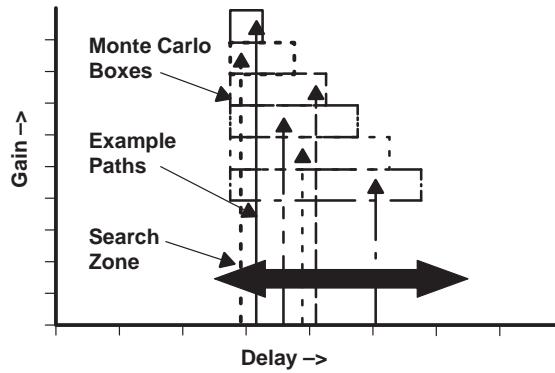


Figure 8. Monte Carlo Model

The Monte Carlo model produces data with delay and path profiles similar to those of the ETSI path model, and at the same time allows a distribution of paths for Monte Carlo modeling.

The ability to decode and receive one of these channels is now measured. The received error rates for 8-bit I/Q data are shown in Table 3.

Table 3. Receive Error Rates

CHANNELS	CAPACITY	EARLY (6 Pilot)	LATE (6 PILOT)	Early (3 PILOT)	LATE (3 PILOT)
48	37.5%	0.2%	0.0%	0.6%	0.1%
64	50%	0.5%	0.2%	2.7%	0.3%
96	75%	3.0%	1.7%	9.6%	1.9%
128	100%	8.1%	5.1%	12.7%	4.3%

Note that all of these errors are correctable by the turbo coding algorithm used to error protect the data channel. The performance of this algorithm is shown in Figure 9. The errors used in the turbo decoding are those produced by the rake receiver.

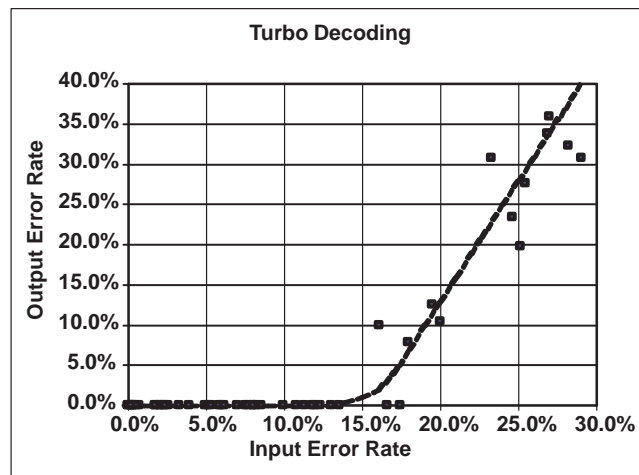


Figure 9. Turbo Decoding Error Correction

Performance is also measured for different A/D resolutions in the radio input. In the simulation environment, it is possible to adjust the input gain to receive valid data with either a 4- or 6-bit A/D converter. In the real world, where the gain is not so controllable and fades must be accommodated, an 8-bit A/D converter is needed. This performance is shown in Figure 10.

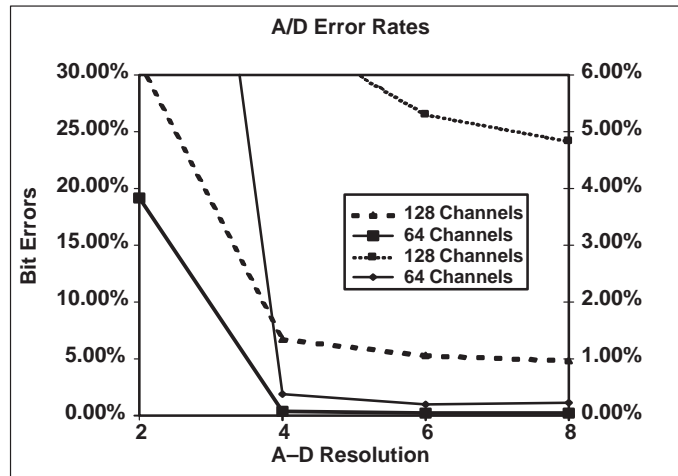


Figure 10. ADC Errors Versus Resolution

5 Loading on the TMS320C62x™ DSP Device

Table 4 details the CPU loading for the various subroutines. Code was compiled with the 28th Dec 3.0.Beta compiler with the `-mt -mh -o3 -mw -k -mx` options. All CPU benchmarks have been performed assuming data and control channels with a spreading factor of 256, giving a data rate of 16 Kbps on each channel. For lower spreading factors, the outer loop overheads are greater; hence the MIP values increase. The full search is the search pattern in which a complete set of new paths is sought for. The quick search is the search pattern in which existing paths are tracked and one new path is found to replace the least significant old path.

Table 4. DSP Cycle Load

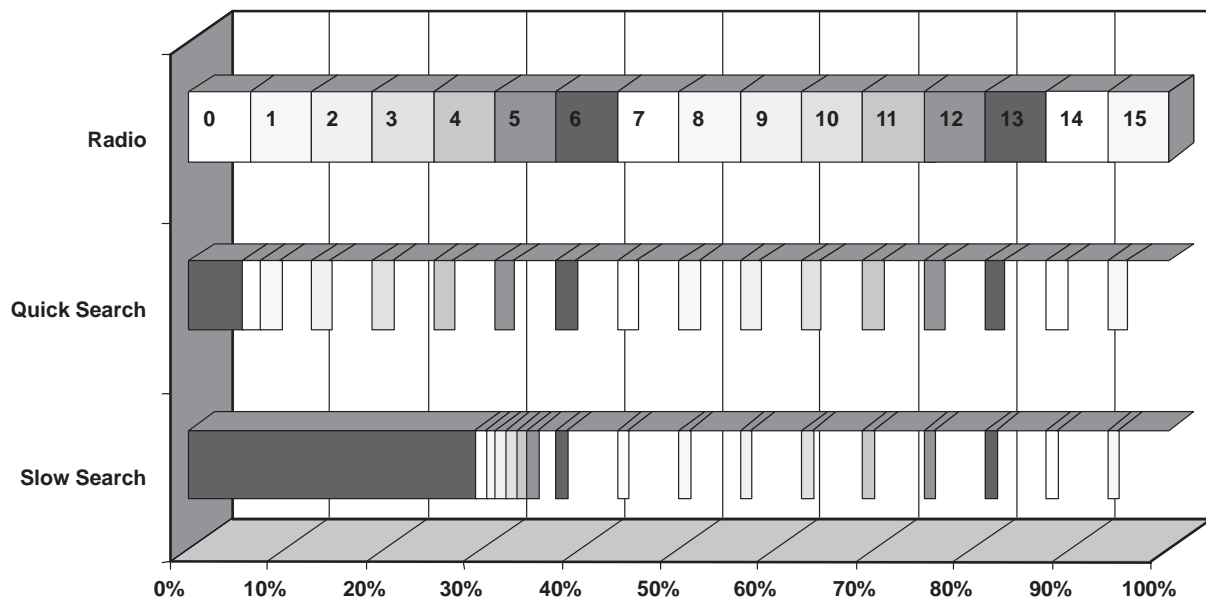
	Pilot(6) Cycles	Pilot(3) Cycles	Repeat	Pilot(6) MHz	Pilot(3) MHz
Turbo	119027		36.5 mS	3.3	3.3
Scramble	2871		0.625 mS	4.6	4.6
Full search	1010278	577818	10 mS	101	57.8
Quick search	205049	112960	10 mS	20.6	11.3
Receive 3 finger	18125		0.625 mS	29	29.0
Receive 6 finger	35349		0.625 mS	56.6	56.6
Totals				137.9 85.1	94.7 75.8

Memory use is shown in Table 5. The main use of memory is in the double buffering of the received radio frames.

Table 5. DSP Memory Load

BYTES	PROGRAM	DATA (ALL)	DATA (1 CH)
Rake	6048	0	8360
Turbo decode	3296	196	0
Scramble	1664	8256	152
Control	4352	189908	0
C-Library	44320	18412	0
Total	59680	216772	8512

However, when real-time CPU loading is done for a 3-bit pilot on a 200 MHz TMS320C62x™ DSP device, a large circular buffer is required, as shown in Figure 11. In fact, it is necessary to buffer 5 full radio frames of I/Q data (or 409,600 bytes) for a 250-MHz device and this is reduced to 4 radio frames.

**Figure 11. Real-Time CPU Load for a 200-MHz TMS320C62x™ DSP Device**

6 Conclusions

The implementation of a software-based WCDMA rake receiver has been shown to be viable on the TMS320C62x™ DSP platform. For a real system, however, multiple users must be detected, which ultimately requires at least one DSP/user. Clearly, this is neither feasible nor competitive as a real solution. It does, however, demonstrate the potential for a DSP and coprocessor implementation, where the majority of the rake operation complexity (that is, the correlation) can be implemented in hardwired form, leaving the more heuristic and intelligent functions in the DSP device.

This application report is based on an early version of the Universal Mobile Telecommunications Systems (UMTS) WCDMA specification and uses real scrambling instead of complex scrambling (complex scrambling doubles the complexity of the correlators). It also does not reflect any of the recommendations made by the Offene Handelgesellschaft (OHG) for 3GPP specification inclusion.

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.