Gold Code Generator Reference Design

March 2003, ver. 1.0 Application Note 295

Introduction

Gold codes are a set of specific sequences found in systems employing spread spectrum or code-division multiple access (CDMA) techniques. These systems are often used in communications equipment such as cellular telephones, global positioning systems (GPS), and Very Small Aperture Satellite Terminals (VSATS). Gold codes have cross-correlation properties necessary in a multi-user environment, where one code must be distinguished from several codes existing in the same spectrum.

This application note describes the implementation of the Gold Code Generator reference design, based on the 3rd Generation Partnership Project (3GPP) specifications for the WCDMA Universal Mobile Telecommunications Systems (UMTS) uplink channel.

Using the time division multiplexing (TDM) technique, the gold code generator is able to generate 32 unique codes simultaneously and efficiently, using the same resources of a single code generator. The design also includes a Nios® embedded processor which handles the tasks of initializing the code generator and switching between different code sets.

Background Information

Pseudorandom Noise Sequences

The pseudorandom noise (PN) sequences are a series of 1's and 0's which lack any definite pattern, and look statistically independent and uniformly distributed. The sequences are deterministic, but exhibit noise properties similar to randomness.

The PN sequence generator is usually made up of shift registers with feedback. By linearly combining elements from taps of the shift register and feeding them back to the input of the generator, you can obtain a sequence of much longer repeat length using the same number of delay elements in the shift register. Hence, these blocks are also referred to as linear feedback shift registers (LFSR).

The length of the shift register, the number of taps, and their positions in the LFSR, are important to generate PN sequences with desirable auto-correlation and cross-correlation properties.

Scrambling Codes in CDMA

Code Division Multiple Access (CDMA) networks allow multiple users to transmit simultaneously within the same wideband radio channel. In order to enable frequency re-use, the networks employ the spread spectrum technique.

The main principle of spread spectrum communication is using wideband, noise-like signals to increase the bandwidth occupancy. As a result of larger bandwidth, the power spectral density is lower, which enables multiple signals to occupy the same band with minimum interference.

During the spreading process, CDMA distributes the signal across the entire allotted frequency spectrum by combining the data signal with a scrambling code which is independent of the transmitted signal. In a multi-path environment, each addressee is assigned a unique scrambling code. The correlation property of these codes makes it possible to generate a distinction between the signals, which allows the different paths to be decoded by the receiver.

Gold Code Generator Functional Description

The scrambling codes used in 3G CDMA wiresless systems are based on "Gold" codes. Gold codes are obtained by combining two PN sequences and modulo-2 adding, or XORing, the output together. These codes have specific cross-correlation properties, to allow as many users as possible, with minimum interference.

Using a set of polynomials, you can construct the PN sequences (also known as m-sequences). This reference design uses a set of specific primitive polynomials over Galois Field 2 (GF[2]) as described in the 3rd Generation Partnership Project (3GPP) Technical Specification 25.213.

The x-sequence uses the following polynomial:

$$X^{25} + X^3 + 1$$

The y-sequence uses the following polynomial:

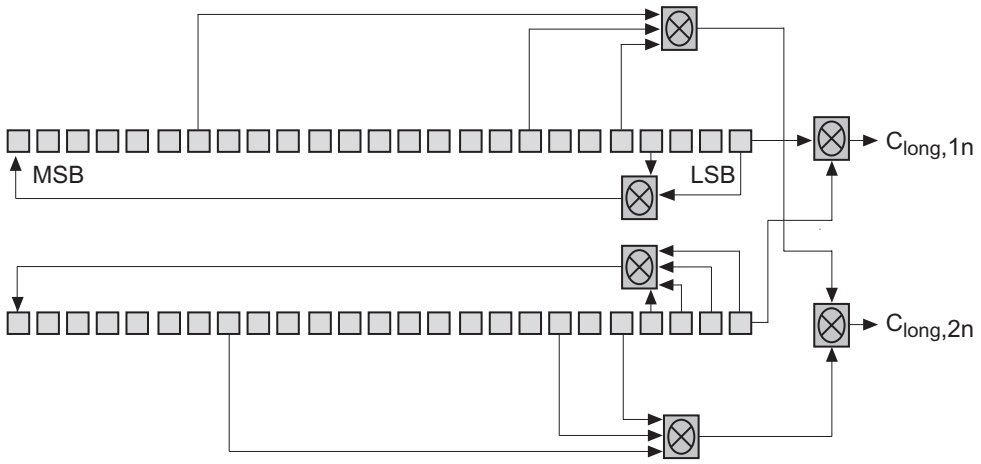
$$X^{25} + X^3 + X^2 + X + 1$$

The specifications require the use of 25-stage LFSRs.

The final result is a long scrambling code, $C_{long,1n}$ and $C_{long,2n}$, generated by summing (using modulo-two addition) the outputs of two PN code sequence generators.

Figure 1 shows the uplink long scrambling code generator block diagram.

Figure 1. Uplink Long Scrambling Code Generator



Reference Design Description

The Altera Gold Code Generator reference design implements a gold code generator targeting the Altera® EP20K400EFC484 device on the APEX™ Nios® development board. You can use this design as a reference for the following.

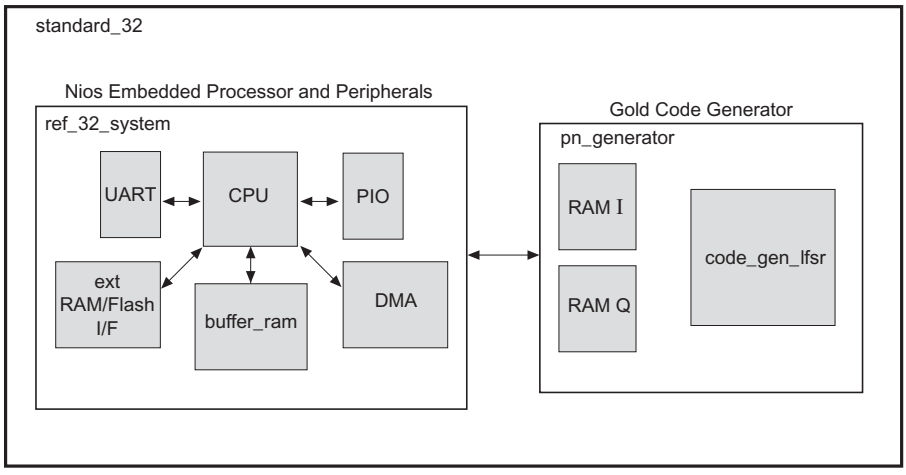
- a scrambling code generator for uplink and downlink channels in 3G CDMA wireless systems
- a spreading signal generator in a Global Positioning System (GPS)

The reference design implements a gold code generator which results in complex-valued long scrambling sequences $C_{long,1n}$ and $C_{long,2n}$. These are separate codes for I and Q. The generation of the sequences is based on an LFSR implementation, as shown in Figure 1. As a result of using the TDM time-sharing technique, the intermediate values of the code need to be stored in two separate RAM blocks.

The reference design also includes a Nios embedded processor to provide an interface to a higher layer in the protocol, which is used to change the initial conditions in the LFSR. The initial condition is also referred to as the "fill" state of the LFSR. By default, the initial values are read from the `init_i.hex`, and `init_q.hex` files.

Figure 2 shows a block diagram of the Gold Code Generator reference design.

Figure 2. Gold Code Generator Reference Design Block Diagram



Port Description

Table 1 describes the ports in the Gold Code Generator reference design.

Table 1. Gold Code Generator Ports (Part 1 of 2)

Port Name	Port Type	Description
Nios CPU and Code Generator		
CLK_DRV_clk_to_apex	In	Clock
RESET_SWITCH_out	In	Reset
Code Generator		
clken	In	Clock enable
stop	In	Stop code generator
Clong_1n	Out	Long scrambling sequences, Clong, 1n
Clong_2n	Out	Long scrambling sequences, Clong, 2n
data_valid	Out	Indicates data on output ports Clong_1n and Clong_2n are valid.
set_code	Out	Indicates end of each time-sharing period (32 clock cycles).

Table 1. Gold Code Generator Ports (Part 2 of 2)

Port Name	Port Type	Description
stop_code_gen	Out	Indicates code generator is stopped. This can be controlled using the “stop” port, or directly from software. Code generator is also stopped during updating of seed values from software.
External RAM and Flash		
SRAM_Hi_data-SRAM_Lo_data[31..0]	In/out	Data to/from off-chip SRAM/Flash
FLASH_high_address_bits-SRAM_Lo_address-FLASH_a0-APEX_a0[19..0]	Out	Address bus to off-chip SRAM/Flash
SRAM_Hi_be_n-SRAM_Lo_be_n[3..0]	Out	Byte enable for off-chip memory
SRAM_Lo_oe_n	Out	Read enable for off-chip memory
SRAM_Hi_cs_n	Out	Chip select for upper 16-bit SRAM
SRAM_Lo_cs_n	Out	Chip select for lower 16-bit SRAM
FLASH_ce_n	Out	Chip select for Flash
FLASH_we_n	Out	Write enable for Flash
SRAM_Lo_we_n	Out	Write enable for SRAM
UART		
DB9_CONNECTOR_rxd	In	RxD pin for UART
DB9_CONNECTOR_rxd_uart2	In	RxD pin for UART (debugger)
DB9_CONNECTOR_txd	Out	TxD pin for UART
DB9_CONNECTOR_txd_uart2	Out	TxD pin for UART (debugger)
Miscellaneous		
FLASH_a16	Out	Flash and RAM chip A17 signals are wired to separate I/O pins for designs using RAM as 16-bit wide device.
HEADER_SWITCH_enable1_n	Out	Enable the switchable, 5V-tolerant proto card pins for the LCD display

The pn_generator Block

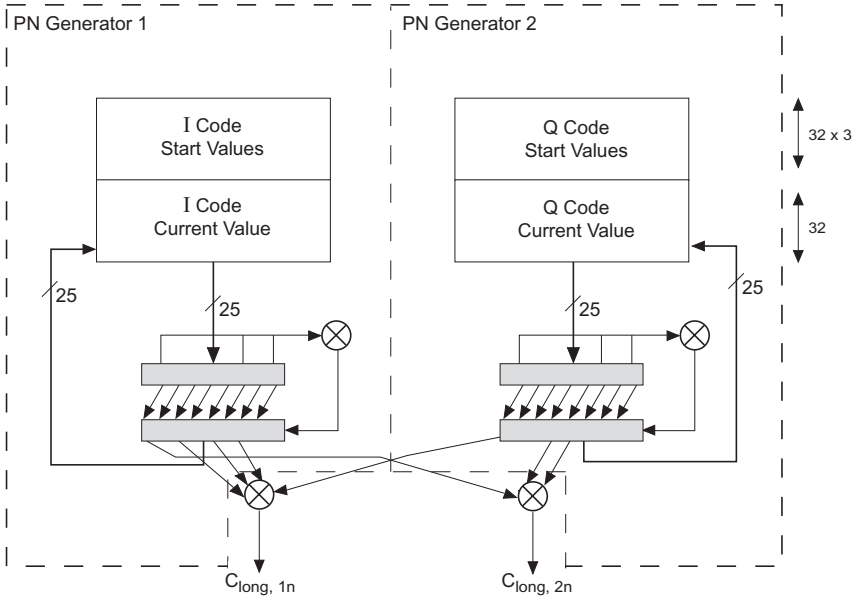
The LFSR in the `code_gen_lfsr` block is implemented using logic cells (LCs). The most significant bits (MSBs) of both the `I` code, and `Q` code are updated based on modulo-2 addition or XOR functions, as shown in Figure 3. Data is read from the respective RAM blocks, processed and written back to memory every clock cycle, to generate 32 separate codes using the time-sharing method. For example, read from address 0, process the code, and write it back to address 0. Next, read from address 1, process the code and write back to address 1, and so on. Upon the final write back to address 31, it cycles back to address 0.

Using a TDM factor of 32, the `pn_generator` block has to run at

32 x 3.84 MHz = 122.88 MHz

in order to support the chip rate of 3.84 Mchip / s (in compliance with the universal terrestrial radio access [UTRA] time division duplex [TDD] specifications in CDMA systems).

Figure 3. Gold Code Generator pn_generator Block



The code generator also has the option of selecting three additional code sets stored in memory by changing the upper bits of the memory address port. Each code set consists of 32 codes. The upper three memory banks are only needed if there is a request from the higher level protocol to stop producing one code, and to insert another. This is controlled directly by the software code via the Parallel Input Output (PIO) peripheral attached to the Nios processor.

The code set values in the RAM blocks can be updated directly if the user decides to change the initialization values in the LFSR. Because of this, the `pn_generator` block also has a multiplexing scheme to switch between the values from the Nios processor and the `code_gen_lfsr` block during write operations to the RAM blocks.

The ref_32_system Block

The `ref_32_system` block describes the Nios system and a set of peripherals. This block provides an interface which emulates message-passing from higher layers in the protocol to the code generator block. This includes changing the initialization values of the LFSR, changing the code sets, and stopping / restarting the code generator.

The initialization values are stored as variables in the software code, `pn_dma.c`. Changing the initialization values of the LFSR involves updating the RAM blocks in the `pn_generator` block. These updates to `ram_blk_i`, and `ram_blk_q` are independent of each other. Using a direct memory access (DMA) block and a buffer RAM block, a set of 32 new values are updated at a time.

The reference design supports four individual code sets, each with 32 codes, resulting in RAM blocks segmented into four separate banks. When making updates to the initialization values of the LFSR, or changing code sets, the user needs to specify the memory bank. This information is passed to the `pn_generator` block via the PIO peripheral. By default, the code generator operates on Memory Bank 1.

During both the changing of the initialization values of the LFSR, and the changing of the code sets, the code generator is stopped. The user also has the ability to stop the code generator directly in hardware, using the `stop` port.

Getting Started

This section describes how to install the Gold Code Generator reference design and walks you through the design flow.

Hardware & Software Requirements

To use the Gold Code Generator reference design, you must have the following software installed on your system.

- The Quartus II software version 2.1, or later
- SOPC Builder version 2.7, or later
- The ModelSim-Altera software version 5.6a, or later

Design Installation

Altera provides the Gold Code Generator reference design as a single, compressed (.zip format) file. To install the files, perform the following steps.

You can download the reference design from the Altera web site at http://www.altera.com.

1. Save the executable file **gold_code_generator.zip** onto your hard disk.

   You can delete this file after you finish installing the design files.

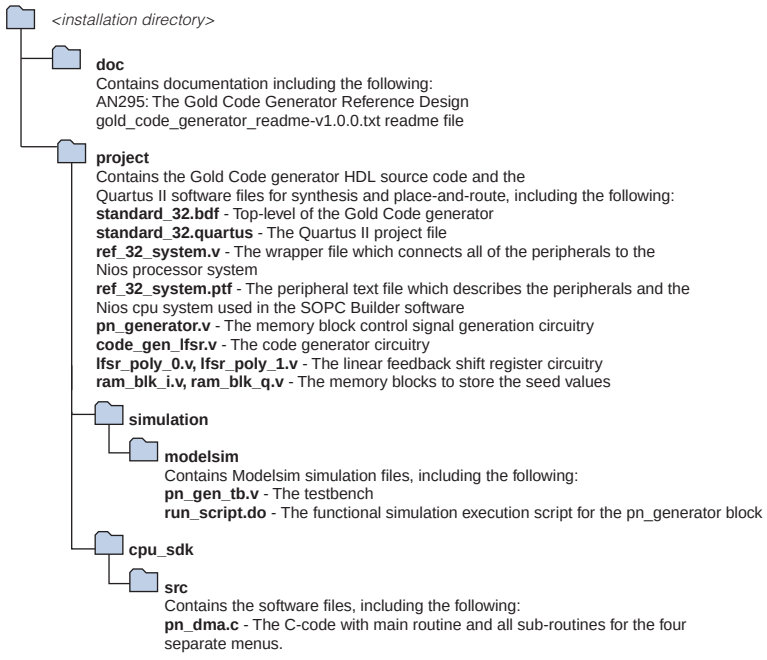
2. Open the Windows Explorer utility, and navigate to the directory in which you have saved the **gold_code_generator.zip** file.

3. Double-click on the **gold_code_generator.zip** file to launch the WinZip™ program.

4. Extract the zipped files to your own installation directory.

Figure 4 shows the directory structure created by the reference design zip file, and describes selected files.

Figure 4. The Directory Structure Created by the Reference Design .zip File



Walkthrough of the Design

Altera provides the source files of the reference design, which you can use to synthesize, place-and-route, and simulate the design. This section walks you through the design flow for the reference design, using these three steps.

1. Compile in the Quartus II software
2. Simulate in the ModelSim-Altera software
3. Run the reference design on the Nios APEX development board

Compile in the Quartus II Software

The <*installation directory*>\project directory contains the Quartus II software version 2.1 project files. These include source files for synthesis and place-and-route within the Quartus II software, and necessary constraint files for the design target (the EP20K200EFC484 device) on the Nios APEX development board.

The following source files are included in the <*installation directory*>\project directory.

- standard_32.bdf
  This is the top-level of the Gold Code generator.
- standard_32.quartus
  This is the Quartus II project file.
- ref_32_system.v
  This is the wrapper file which connects all the peripherals to the Nios processor system.
- ref_32_system.ptf
  This is the peripheral text file which describes the peripherals and the Nios processor system used in the SOPC Builder software.
- pn_generator.v
  This is the memory block control signal-generation circuitry.
- code_gen_lfsr.v
  This is the code generator circuitry.
- lfsr_poly_0.v, lfsr_poly_1.v
  This is the linear feedback shift register circuitry.
- ram_blk_i.v, ram_blk_q.v
  These are the memory blocks that store the seed values.

To compile the Altera-provided project files, follow these steps.

1. Run the Quartus II software.
2. Choose **File** > **Open Project**.
3. Browse to the <*installation directory*>\project directory.
4. Select the project file `standard_32.quartus` file and click **Open**.
5. Choose **File** > **Open** and select the `standard_32.bdf` file and click **Open.**

6. Double-click on the `ref_32_system` symbol to bring up the SOPC Builder software.

7. Click on the **Generate** button to generate all of the Nios processor-related files. When all of the files are generated, click on **Exit** to close the SOPC Builder software.

8. Choose **Processing** > **Compile Mode**.

9. Choose **Processing** > **Compile Settings** and select `standard_32` as the Compilation focus in the **General** tab.

10. Click on OK.

11. Choose **Processing** > **Start Compilation**.

Simulate in the ModelSim-Altera Software

Prior to running the behavioral simulation in ModelSim, it is necessary to change the path settings in the <*installation directory*> **\project\simulation\modelsim\run_script.do** file to point to the location of the files, and the ModelSim-Altera software, using the "path_name" and "modelsim_path" variables, respectively.

To perform behavioral simulation with the ModelSim software, perform the following steps.

1. Start the ModelSim-Altera software.

2. Choose **File >Change Directory**.

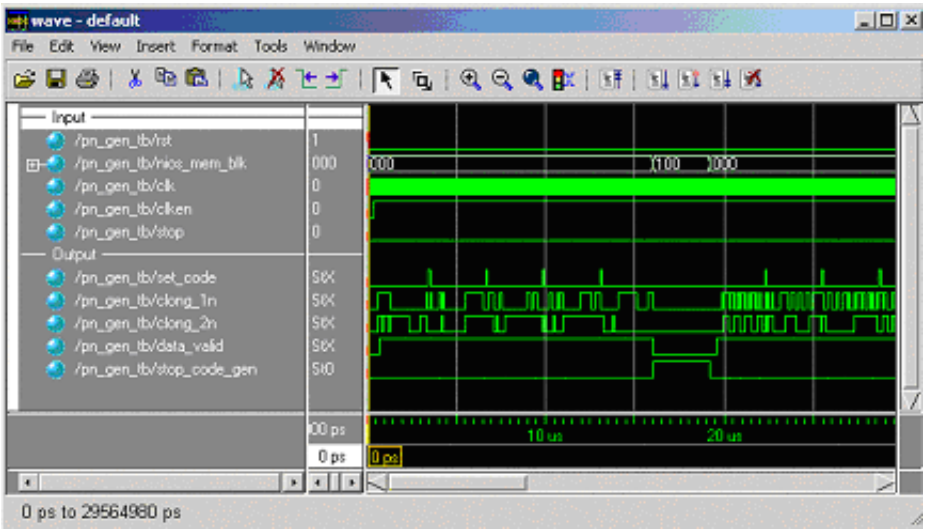
3. Browse to the <*installation directory*> **\project\simulation\modelsim** directory and click **Open**.

4. Choose **Macro > Execute Macro**.

5. Browse to the `run_script.do` script, and click **Open**.

The simulation results are displayed in a waveform as shown in Figure 5. The testbench initializes all of the design registers and simulates the write operations from the Nios embedded processor when updates are made to the seed values in the RAM blocks within the pn_generator block.

Figure 5. Behavioral Simulation of the pn_generator Block in the ModelSim Software



Run the Design on the Nios APEX Development Board

Connect the Cables to the Board

Before running the design on the APEX Nios development board, you must connect the cables to the board.

Refer to the *Nios Embedded Processor Getting Started User Guide* for more details on setting up the development board. Refer to the *Nios Embedded Processor Development Board Data Sheet* for more details on the board itself.

Perform the following steps to connect the cables.

1. Connect the power adapter cable to the board and plug it into a 110V AC power outlet.
2. Connect the ByteBlasterMV™ cable between your machine and the the board's 10-pin JTAG header for APEX configuration.
3. Connect an RS-232 cable to your machine and to the board.

Configure the APEX Device

Perform the following steps to configure the APEX device.

1. Run the Quartus II software.
2. Choose **Processing > Open Programmer**.
3. Click on the **Add File** button.
4. Browse to the **<*installation directory*>\project** directory.
5. Select the file `standard_32.sof` and click on **Open**.
6. Turn on the **Program/Configure** option.
7. Click **Start** to configure the APEX device.

Compile and Download the Reference Design Executable File

Before you download the software executable, you need to compile the C-code `pn_dma.c` file, using the GNU Pro Compiler.

Refer to the *Nios Embedded Processor Software Development Reference Manual* for details on the commands.

Perform the following steps to compile and download the software executable.

1. Run the Nios SDK Shell.

   The shell window appears and displays a shell prompt.

2. Navigate to the **<*installation directory*>\project\cpu_sdk\src** directory.

3. Type `nios-build pn_dma.c` <*return*> at the Nios SDK Shell prompt.

   The Nios build utility will invoke the compiler and linker and produce several intermediate files, and an executable (.srec) file.

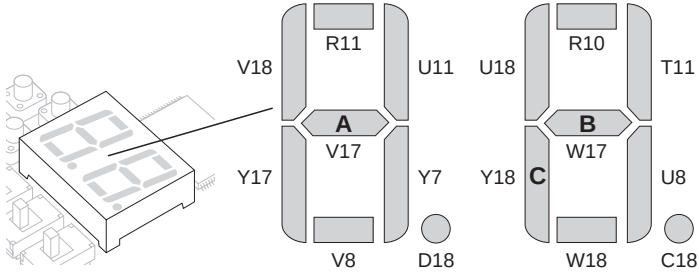
4. Type `nios-run pna_dma.srec` <*return*> to download the srec file over the serial port and begin execution.

   Your Nios system is now running, and you will see the Main Menu, as shown in Figure 6.

Figure 6. Nios Processor Main Menu

The `clken` and `stop` input signals are controlled by DIP_SWITCH_1 and DIP_SWITCH_2, respectively. The `stop_code_gen`, `set_code`, and `data_valid` output signals can be viewed on the 2-digit seven-segment display on the board, as shown in Figure 7.

Figure 7. The 2-Digit, Seven-Segment Display



A: stop_code_gen
B: set_code
C: data_valid

The `clong_1n`, and `clong_2n` scrambling codes are connected to LED1 and LED2, respectively.

Resource Usage

For APEX devices, the Gold Code Generator reference design requires the device resources shown in Table 2.

Table 2. Required APEX Device Resources

Logic Cells	Memory Bits	I/O Pins
4351	41088	77

Table 3 shows the resource usage of the two main blocks in the Gold Code Generator reference design.

Table 3. Main Blocks Resource Usage

Block Name	Logic Cells	Memory Bits
ref_32_system	4094	34688
pn_generator	257	6400

Support

For information or support for the Gold Code Generator reference design, go to *http://mysupport.altera.com* or contact Altera Applications.

Conclusion

The Gold Code Generator reference design is efficient because the implementation allows time sharing. In reusing the same resources, the code generator is able to produce 32 separate codes simultaneously. The design also works at a sub-system level where the code generator is able to interact with a higher layer in the protocol. This is achieved using the Nios embedded processor where it provides the interface to transfer high-level messages to the code generator (to change the initialization values of the LFSR, and the selection of the code sets). This reference design presents a building block that will help shorten the development time of your W-CDMA system.

Notes: