Binary Pattern Correlation and Frame Synchronization

By Xiaofei Dong
Ai Hua
Gautam Karnik
CDMA Based Communication System Group

Application

In a digital communication system or a digital signal processing application, frame synchronization is one of the most fundamental issues in the system design. Usually, the control information and data have been assembled into a frame from the network layer, and the synchronization code is attached at the start of a frame at the data link layer. The complete frame is sent to physical layer for spreading (in a spreading communication system) and modulation. Correct synchronization at the receiver is the basis of the functional system.

The burst communication system includes time division multiplexing (TDM), code division multiplexing (CDMA) or the simpler time division dulplexing (TDD). In such a system, each user is assigned a time slot for transmission and multiple user data are collected by a central processing node. A frame is assembled and with certain binary pattern as the frame synchronization code. The receiver searches this binary pattern to determine the transmitter timing as well as the start of the frame data. In such a system, a binary pattern correlator is needed at the receiver for

- Matching the binary pattern
- Achieve frame synchronization
- Transmitter timing
- Valid frame data for the next module to process.

Once the frame synchronization pattern is established, the frame boundary as well as the receiver frame timing is aligned with the transmitter. This pattern correlator can also be used as despreader in a spreading communication system.

As part of Altera Megafunction Partner Program, NOVA Engineering Inc. developed a product called "Binary Pattern Correlator" [1] to implement the above-mentioned functions. Instead of purchasing from NOVA, the detailed design as well source code of such a binary pattern correlator is introduced in this application note. The design has been simulated and tested in our CDMA based communication system project and it can achieve all the features mentioned below.

Features:

- Designed for Altera FLEX 10K20RC240-4 FPGA
- Parallel processing and pipelined summing network to achieve maximum speed
- Programmable reference pattern (synchronization code)
- Application includes:
    - Frame synchronization
    - Pattern recognition
    - Despreading in a spreading communication system

General Description

The synchronizer introduced here is a binary pattern correlator, which compares the input digital data with the preset synchronization code (binary pattern). The correlator contains:

- A shift register that converts the serial input bits into parallel data
- Synchronization code (reference pattern) register which stores the synchronization code
- Correlation unit that correlates the received data with the reference pattern
- Summing network which finds the correlation of the input data and the reference code
- Decision device that makes the decision of which logic data have been received according to the result of the summing network.

Functional Description

To understand the design of the binary pattern correlator, some technical terms have to be explained. We define the distance of two signals or two sequences as the number of term-by-term disagreement. Then the correlation of the two sequences should be the number of term-by-term agreement [2]. For example, for signals (in the digital communication a signal is a sequence of logic numbers) A and B where

$A= \{a_1, a_2, \ldots a_n\}$ $B=\{b_1, b_2, \ldots b_n\}$
Correlation{A, B} = N- distance{A, B} i=1, 2, …n.

First the bit stream coming in the pattern correlator is shifted into a shift register and compared with the preset reference pattern. The comparison is the negated exclusive or logic operation on the two input vectors. We denote the output of the comparison as the weight vector. The number of '1's in the weight vector is the correlation of the two sequences. The correlation is obtained from the summing network. Then at the decision unit the correlation is compared with a threshold to make a decision whether a synchronization code has been received. For example, if a 32-bit synchronization code is used, the threshold can be set to (>= 30). The threshold of 32 requires a complete match, while a lower threshold can give some margin when the frame suffers from noise and interference. The bottleneck of the design is how fast the correlation of the signals can be found. An adder is one choice, but for synchronization code size larger than 4, the delay in finding the distance of two sequences is almost intolerable for the design of the digital system. We provide a pipelined summing network with reduced logic complexity of the summing operation.

Ports

Table 1 describes the ports for the synchronizer.

Table 1. Synchronizer Ports

Name	**Type**	**Size (bits)**	**Description**
Rx	Input	1	Serial or sampled input
Sync_code	Input	Variable; 32 in our design	Synchronization code or reference pattern
device_en	Input	1	System work when set to high
Clock	Input	1	System clock
Reset	Input	1	System reset

Table 1. Synchronizer Ports			
frame_bit	Output	1	Frame data in serial output; synchronization code has been stripped from the frame data
bit_ready	Output	1	Flag signal of detecting the sync code

Parameters

All the register length and the reference code are implemented as generic data so can be easily modified.

Table 2. Synchronizer Parameters		
Name	**Typical value**	**Customization**
serial_to_parallel register length	8 to 32(32 used in our design)	Changeable length
sync_code (reference pattern)	Lengh: 8 to 32(32 used in our design)	Changeable length and value
Correlation sum	6(for 32 code length)	Designed for 32 bits code length

Performance

The bottleneck of the design for this synchronizer is the summing network, which finds the number of '1's in the weight vector. By using pipelines and reduced adder structure, the synchronizer can operate at a high speed (much better than the Binary Pattern Correlator of NOVA that has maximum clock frequency of 40Mhz according to their data sheet).

Table 3. Typical Device Utilization (using FLEX10K20RC240-4)			
Implementation	Clock (f_{max})	Logic Cells	EABs
Input data width =1 bit Serial-to-parallel register width = 32 bits Synchronization code length = 32 bits Pipeline stages = 4	59.52 MHz	63 (For using sync_code as a constant)	0

Implementation

The key design of the correlation is the summing network, which must operate at the high speed. Suppose we use a 32-bit synchronization code, the counting of the ‘1’s in the weight vector gives a value between 0 and 32. So the output of the summing network is a 6-bit vector.

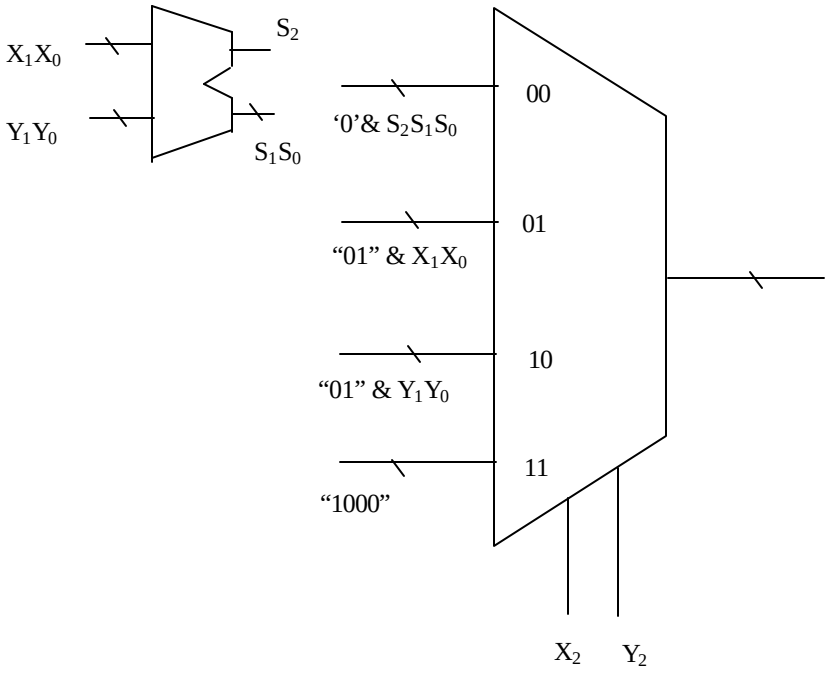
The weight vector obtained from the correlating unit indicates whether the two sequences are the same at the each position. We need to go through every bit to count how many term-by-term agreements they have.

We group every two adjacent bits and map them to the inputs of 16 half adders. Then we take the output of each half adder as a 2-bit vector. The carry is the MSB; the sum is the LSB. Thus the outputs of the half-adder bank are 16 2-bit vectors. Then we need to sum all of the 16 vectors. Eight 2-bit ripple carry adders can be used to do the first stage summation. But we notice that the input to the 2-bit adder can never be "11". That is, not all combinations of the inputs are possible at the input to the adder. Again, we let the carry output be the MSB of the output vector of each adder, thus form a vector length of 24 bits in all.

The second stage summation can be done by four 3-bit adders, again the input to the adder will be no larger than b"100" since there can be no more than four '1's in a 4-bit binary vector. We can thus design an incomplete adder, which reduces the n-bit ripple carry adder into a (n-1)-bit adder in certain situation. After that, two 4-bit adders and one 5-bit adder are needed to complete the summation.

The incomplete 3-bit adder I propose here is like a 4-to-1 multiplexer. For example, suppose the input addends to a 3-bit incomplete adders are { $x_2 x_1 x_0$} and {$y_2 y_1 y_0$}. { $x_2 x_1 x_0$} are the outputs from a 2-bit incomplete adder of the previous stage. x_2 is the carry and $x_1 x_0$ are the sum. They show how many ones are in the 4-bit section of the weight vector. Thus, the possible combination of { $x_2 x_1 x_0$} are X"0" to X"4". We can use the MSB of vectors x and y as the selector of the multiplexer. When $x_2 y_2$ = "00", lower bits of x and y sum like a conventional 2-bit ripple carry adder, and the carry of the 3-bit incomplete adder is zero. When $x_2 y_2$ = "01" or "10", value 4 is summed with a vector that is less than 4, so the output carry of the adder is zero, the most significant of the sum part is '1', and the lower bits of the sum part is just the addend that is not four. When $x_2 y_2$ = "11", all other bits of x and y must be zeros, so the output is eight. As a summary of the above explanation, the schematic of the incomplete adder is as follows:

Fig. 1 Schematic of the incomplete 3-bit adder



Similarly, the incomplete 2-bit, 4-bit and 5-bit adders are obtained by giving different values to the generic adder size.

This design allows the counting of the '1's in a 32-bit vector to operate at the speed of 21.4MHz (maximal clock frequency). To speed up the operation, pipeline registers can be inserted after the adder bank. The pipeline register width of the first, second, third and fourth stage is 32, 24, 16 and 10 respectively. When the tradeoff the speed and resources need to be considered, pipeline stages can be inserted only after the third or the fourth stage where the summation of the ripple carry adder is slow. When four pipeline stages are used, the maximal clock frequency is found to be 59.52 MHz. In the application of Altera UP-1 board, where the system clock frequency is 25MHz, this frequency leaves enough time margins for preparing data at the output.

When pipeline registers are used, the frame data after the synchronization code need to be delayed by corresponding clock periods in the synchronizer. This can be simply achieved by sending the corresponding serial-to-parallel register bit as the output bit.

NOTE:
To save resources, the synchronization code or the reference pattern can be set as constants in VHDL instead of using a register. The number of logic cells saved by using constants varies with implementation of other parts of the module and the synchronization code or pattern chosen.

References

1. Altera Binary Pattern Correlator Megafunction, Solution Brief, April 1997, ver. 1
2. J.S. Lee, L. E. Miller, "CDMA Systems Engineering Handbook", Artech House: Boston, 1998.