Proceedings of the 10th WSEAS International Conference on CIRCUITS, Vouliagmeni, Athens, Greece, July 10-12, 2006 (pp41-44)

The Modulo-10 Partition Counter

Dr. SERAFIM PORIAZIS
Phasetronic Laboratories
6 Depasta Str., 17122, N. Smirni, Athens
GREECE
http://www.phasetroniclab.com

Abstract: − The objective of the paper is to design a Modulo-10 Partition Counter as a module written in VHDL. This module generates a repeating sequence of fifty-two partitions under the control of the input clock signal. Each partition is expressed by an array of appropriate integer values based on the mathematical Restricted Growth String notation. A phase difference equal to the half period of the clock signal is used internally to achieve the correct pulse timing of the output. The output signal patterns correspond to the blocks of each partition in a phase encoded format for a duration of ten phases. The VHDL description of the MPC10 module is given and the simulation and synthesis results are presented.

Key-Words: − block, clock, counter, design, partition, phase, VHDL.

1 Introduction

Counter-based exhaustive testing utilize binary counters of size equal to the number of compatibility classes, i.e., the test length is 2^N where N is the number of compatibility classes. A Pk-compatibility relation is proposed by [1], where a partition b_i is defined on the set of compatibility classes with blocks consisting of one or more compatibility classes. The test length of the counter-based exhaustive testing can be reduced by using the Pk-compatibility relation as being equal to 2^S where S is the number of blocks for each partition b_i.

The Counter (CTR) Mode of operation is to be used with the Advanced Encryption Standard (AES). The sequence of counter blocks must have the property that each block is different from others while the same given key is used and this is accomplished by an *incrementing function*. The use of a Linear Feedback Shift Register (LFSR) is recommended to avoid the vulnerability of a single or multiple bit faults based on the presented fault model of the counter mode [2]. One crucial performance challenge of memory encryption is the decryption latency. Fast protection schemes based on counter mode allows parallel execution of encrypted data fetching and decryption pad generation. A counter (also called *sequence number*) associated with each data block has to be cached inside the secure processor to exploit such parallelism enabled by counter mode. The proposed technique by [3] hides the latency overhead of decrypting counter mode encrypted memory by predicting the sequence number and pre-computing the encryption pad that is called *one-time-pad* or OTP. The concept of counter mode in general and its application to secure processor design is also discussed.

Starting the counter in an arbitrary state may result in a transient period until the counter reaches a state that is part of the periodic sequence, which is exactly what happens to the up counter when started in the zero state [4]. For any systolic counter we can find the sequence of 2^N states that are part of the periodic sequence by starting the counter in a state that has a unique representation. The given counter enters a repetitive sequence directly after reset as it has a unique representation of the zero state.

In this paper we consider the design of a counter that produces a periodic sequence of partitions. In particular, the design of the Modulo-10 Partition Counter (MPC10) is given, which utilizes a frame of 10 phases of the input clock signal and outputs 1-out of 52 partitions per frame. This module assists the reliability of operation of the multiphase model [5] whose operation adopts a 10-phase timing pattern and is targeting data streaming applications. The blocks of each partition are presented by the five output signals in an encoded format. The VHDL description of the MPC10, the corresponding simulation and synthesis results targeting an FPGA device are given.

2 The MPC10 module

The fundamental module, which generates a repeating sequence of fifty-two (52) partitions under

the control of the clock signal CLK of frequency f, each counting value lasting ten (10) halves of the clock period, is called the Modulo-10 Partition Counter (MPC10). The module block diagram and its VHDL description are shown in Figures 1 and 2 respectively. Considering the half clock period as being a clock phase, we use a ten phase frame during which the counting value is presented as a pulse pattern corresponding to 1-out-of-52 partitions. The expression modulo-10 is used to signify that the counter produces its counting value (one partition) during this ten-phase frame. The fifty-two partitions have a counting index K in the range from 0 to 51. The content of each block of a partition of this module is encoded by using the mathematical Restricted Growth String notation for N=5, thus requiring five output lines. For example, having the counting index K=4, we have a three block partition with blocks B_0=(output-line-0, output-line-1, output-line-2), B_1=(output-line-3) and B_2=(output-line-4) where the pulse timing of the first block holds the 1st phase, the second block holds the 3rd phase, and the third block holds the 5th phase of the frame of the input clock signal. The above partition example is encoded by the Restricted Growth String notation as the sequence {0,0,0,1,2} where the element 0 of this sequence signifies block B_0, the element 1 block B_1 and the element 2 block B_2. Similarly holds for each one of the remaining partitions. All encoded partitions have been sorted in ascending order in order to define the counting index K (from 0 to 51) as shown in Table 1. We note that we use an input clock signal with a duty cycle of 50 percent, thus having a pulse of logic-1 or logic-0 value for a half period or a phase.

3 The VHDL simulation and synthesis

The VHDL testbench simulation results for the MPC10 module are given in Figure 3. The duration of this simulation is defined by the value of the signal "done". The internal signal *pclk* has a width of 10 bits. The output port GCLK is analyzed into five individual output signals with waveforms that verify the correct operation of the module (for demonstration purposes only the counter values *ptn*/k1 of "0,0,0,0,0"/0, "0,0,0,0,1"/1, "0,0,0,1,0"/2, "0,0,0,1,1"/3, "0,0,0,1,2"/4 and "0,0,1,0,0"/5 are shown, each for a duration of one frame). The logic value changes of the internal phased signals *pclk* occur at each rising and at each falling edge of the input signal CLK.

Table 1. The MPC10 output values based on the mathematical Restricted Growth String notation

Count K	partition	Count K	partition
0	00000	26	01101
1	00001	27	01102
2	00010	28	01110
3	00011	29	01111
4	00012	30	01112
5	00100	31	01120
6	00101	32	01121
7	00102	33	01122
8	00110	34	01123
9	00111	35	01200
10	00112	36	01201
11	00120	37	01202
12	00121	38	01203
13	00122	39	01210
14	00123	40	01211
15	01000	41	01212
16	01001	42	01213
17	01002	43	01220
18	01010	44	01221
19	01011	45	01222
20	01012	46	01223
21	01020	47	01230
22	01021	48	01231
23	01022	49	01232
24	01023	50	01233
25	01100	51	01234

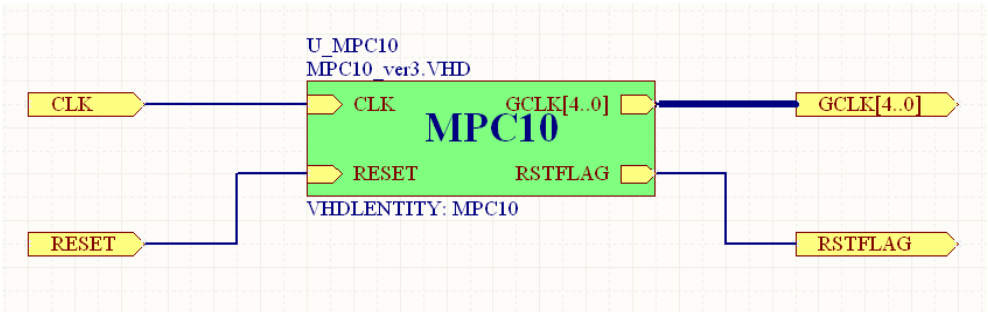

Fig. 1. The MPC10 block diagram

```
library IEEE;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_1164.all;

entity MPC10 is
    port (CLK , RESET : in std_logic := '0';
          RSTFLAG : out std_logic;
          GCLK : out std_logic_vector(4 downto 0) );
end MPC10;

architecture behavioral of MPC10 is
      type RG_string is array(0 to 4) of integer;
      type validpartitions is array(0 to 51) of RG_string;
      type validcodewords is array(1 to 20) of std_logic_vector(10 downto 1);
      constant phased_output : validcodewords :=
      (  ('0','0','0','0','0','0','0','0','0','0'), ('0','0','0','0','0','0','0','0','0','1'),
         ('0','0','0','0','0','0','0','0','1','1'), ('0','0','0','0','0','0','0','1','1','1'),
         ('0','0','0','0','0','0','1','1','1','1'), ('0','0','0','0','0','1','1','1','1','1'),
         ('0','0','0','0','1','1','1','1','1','1'), ('0','0','0','1','1','1','1','1','1','1'),
         ('0','0','1','1','1','1','1','1','1','1'), ('0','1','1','1','1','1','1','1','1','1'),
         ('1','1','1','1','1','1','1','1','1','1'), ('1','1','1','1','1','1','1','1','1','0'),
         ('1','1','1','1','1','1','1','1','0','0'), ('1','1','1','1','1','1','1','0','0','0'),
         ('1','1','1','1','1','1','0','0','0','0'), ('1','1','1','1','1','0','0','0','0','0'),
         ('1','1','1','1','0','0','0','0','0','0'), ('1','1','1','0','0','0','0','0','0','0'),
         ('1','1','0','0','0','0','0','0','0','0'), ('1','0','0','0','0','0','0','0','0','0') );
       constant partition : validpartitions :=
       ( (0,0,0,0,0), (0,0,0,0,1), (0,0,0,1,0), (0,0,0,1,1), (0,0,0,1,2), (0,0,1,0,0),
         (0,0,1,0,1), (0,0,1,0,2), (0,0,1,1,0), (0,0,1,1,1), (0,0,1,1,2), (0,0,1,2,0),
         (0,0,1,2,1), (0,0,1,2,2), (0,0,1,2,3), (0,1,0,0,0), (0,1,0,0,1), (0,1,0,0,2),
         (0,1,0,1,0), (0,1,0,1,1), (0,1,0,1,2), (0,1,0,2,0), (0,1,0,2,1), (0,1,0,2,2),
         (0,1,0,2,3), (0,1,1,0,0), (0,1,1,0,1), (0,1,1,0,2), (0,1,1,1,0), (0,1,1,1,1),
         (0,1,1,1,2), (0,1,1,2,0), (0,1,1,2,1), (0,1,1,2,2), (0,1,1,2,3), (0,1,2,0,0),
         (0,1,2,0,1), (0,1,2,0,2), (0,1,2,0,3), (0,1,2,1,0), (0,1,2,1,1), (0,1,2,1,2),
         (0,1,2,1,3), (0,1,2,2,0), (0,1,2,2,1), (0,1,2,2,2), (0,1,2,2,3), (0,1,2,3,0),
         (0,1,2,3,1), (0,1,2,3,2), (0,1,2,3,3), (0,1,2,3,4) );
        signal index, k1, k2, kn, m : integer := 0;
        signal present_state1, present_state2, next_state, pclk: std_logic_vector(10 downto 1);
        signal invalidcode_flag : std_logic := '0';
      begin
       reg1a : process (CLK, RESET)
          begin
             if  RESET = '1' then  present_state1 <= phased_output(1);
             elsif (CLK='1' and CLK'event )  then present_state1 <= next_state ;
             end if;
       end process;
       reg1b : process (CLK, RESET)
          begin
             if  RESET = '1' then  present_state2 <= phased_output(11);
             elsif (CLK='0' and CLK'event )  then present_state2 <= next_state ;
             end if;
       end process;
        reg2a : process (CLK, RESET)
          begin
             if  RESET = '1' then  k1 <= 0;
             elsif (CLK='1' and CLK'event )  then k1 <= kn ;
             end if;
       end process;
       reg2b : process (CLK, RESET)
          begin
             if  RESET = '1' then  k2 <= 0;
             elsif (CLK='0' and CLK'event )  then k2 <= kn ;
             end if;
       end process;
       next_state_logic : process (CLK, RESET)
          begin
             case CLK is
              when '1' =>  if RESET = '1' then index <= 1;  kn <= 0; m <= 0;
                           else index <= index + 1; m <= m+1; end if;
              when '0' =>  if RESET = '1' then index <= 11; kn <= 0; m <= 0;
                           else index <= index + 1; m <= m+1; end if;
              when others   => null;
             end case;
             if m >= 10*52 then m <= 1; end if;
             if m >= 1 and m <= 10*52 and (m mod 10) = 0 then kn <= kn+1; end if;
             if m >= 10*52 or kn >= 52 then kn <= 0; end if;
             if index < 20 then next_state <= phased_output(index + 1);
             else next_state <=  phased_output(1); index <= 1;  end if;
             for i in 1 to 20 loop
               if next_state = phased_output(i) then invalidcode_flag <= '0'; exit;
               else invalidcode_flag <= '1'; end if;
             end loop;
        end process;
        output_logic : process (index, present_state1, present_state2)
          variable ptn : RG_string;
          begin
            case CLK is
              when '1' =>  pclk <= present_state1; ptn := partition(k1 mod 52);
                           if RESET = '1' then RSTFLAG <= '1'; else
                           RSTFLAG <= invalidcode_flag;  end if;
              when '0' =>  pclk <= present_state2; ptn := partition(k2 mod 52);
                           if RESET = '1' then RSTFLAG <= '1'; else
                           RSTFLAG <= invalidcode_flag;  end if;
              when others   => null;
            end case;
            for n in 0 to 4 loop
             case ptn(n) is
              when 0 => GCLK(n) <= pclk(1) xor pclk(2);
              when 1 => GCLK(n) <= pclk(3) xor pclk(4);
              when 2 => GCLK(n) <= pclk(5) xor pclk(6);
              when 3 => GCLK(n) <= pclk(7) xor pclk(8);
              when 4 => GCLK(n) <= pclk(9) xor pclk(10);
              when others => GCLK(n) <= '0';
             end case;
            end loop;
          end process;

end behavioral;
```

Fig. 2. The VHDL description of the MPC10 module

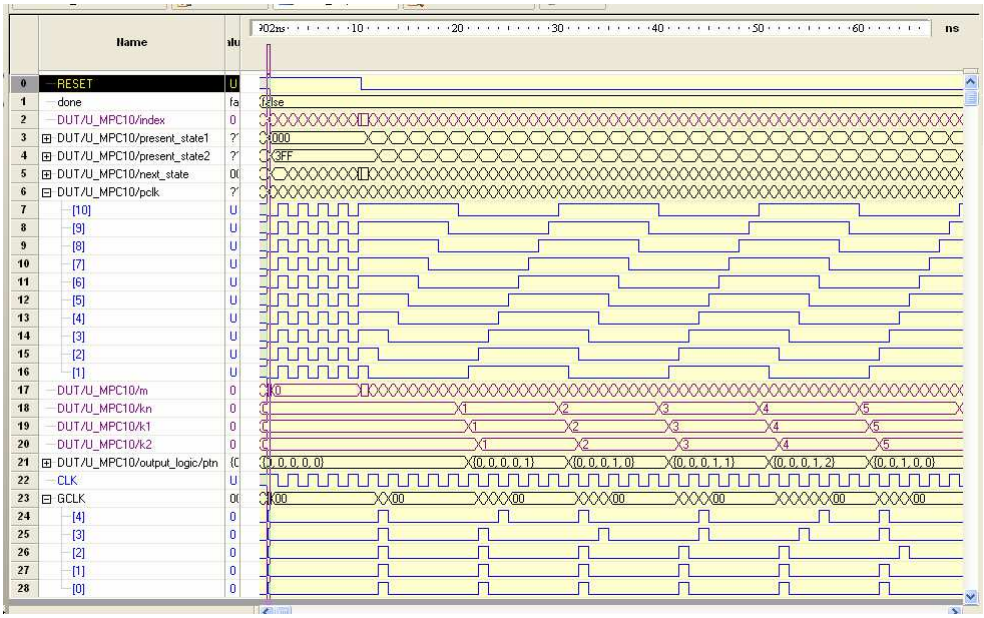

Fig. 3. The MPC10 operation (VHDL simulation results)

The synthesis of the MPC10 module targeting an FPGA device was successfully performed giving us the following results:

- flip flops with asynchronous reset = 74
- flip flops with asynchronous preset = 10
- combinational feedback paths = 96
- combinational logic area estimate = 3432 LUTs

4 Conclusion

The design aspects of the Modulo-10 Partition Counter (MPC10) module written in VHDL are being considered in this paper. The operation of this module is based on a set of internal phased signals that are used to generate a repeating sequence of timing patterns, that is, the fifty-two partitions under the control of the input clock signal by utilizing a frame of 10 phases.

The VHDL description of the MPC10 module is given. The corresponding simulation results verify the proper circuit operation while the internal phased signals *pclk*[10..1] and the output signals GCLK[4..0] maintain the phase associations and the counter value specification being expressed by the mathematical Restricted Growth String notation. The synthesis results of the MPC10 are given targeting an FPGA device.

References:

[1] E. Gizdarski, and H. Fujiwara, "Fault set partition for efficient width compression", in *Proceedings of the 11th Asian Test Symposium, 2002 (ATS '02)*, 18-20 Nov. 2002, pp.194-199.

[2] R. Tirtea, and G. Deconinck, "Specifications overview for counter mode of operation. Security aspects in case of faults", in *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference, 2004 (MELECON 2004)*,12-15 May 2004, Vol.2, pp.769-773.

[3] Shi.Weidong, H.S.Lee, M.Ghosh, Lu.Chenqhuai, and A.Boldyreva, "High efficiency counter mode security architecture via prediction and precomputation", in *Proceedings of the 32nd International Symposium on Computer Architecture, 2005 (ISCA '05)*, 4-8 June 2005, pp.14-24.

[4] M.R.Stan, "Systolic counters with unique zero state", in *Proceedings of the 2004 International Symposium on Circuits and Systems(ISCAS '04)*, 23-26 May 2004, Vol.2, pp.II-909-12.

[5] S. Poriazis, *Logic Design of Multiphase Finite State Machines*, Monograph, Phasetronic Laboratories, http://www.phasetroniclab.com Athens, Greece, 2001.

[6] S. Poriazis, "The Two-Phase Twisted-Ring Counter Circuit", in *Proceedings of the 2002 IEEE International Symposium on Circuits and Systems, ISCAS'02*, Phoenix, Arizona, USA, vol. IV, 26-29 May 2002, pp. 858-861.