Implementation of Golay Complementary Code Sequences Generator Based on FPGA

Dhamyaa H. Mohammed
Computer & Communication Engineering, Al-Mansour University College/ Baghdad
Email: dhamyaa.Husam@yahoo.com

Received on: 18/10/2012 & Accepted on: 4/4/2013

ABSTRACT
Golay sequences have some properties make it distinctive in the applications and results. However, for this distinction must select the code sequences carefully and accurately. Therefore, to satisfy these requirements, a creation algorithm must be easy, accurate and powerful. In this paper, an FPGA based, design and implementation of Golay complementary code sequence (GCCS) creation and then made autocorrelation between their pair codes to verify properties. The process time for proposed algorithm is less than that for all possible algorithm by (1/4 to 1/1024 for 4-bit to 16 bits respectively). Thus, the Search can be regarded as pioneers of the research application of this technique to the subject and got good results. The Implementation was based on 8-bit pair code and made by Xilinx-spartan-3A XC3S700A FPGA, with 50 MHz internal clock.

Keywords: Golay complementary sequences, FPGA-Spartan and correlator.

INTRODUCTION
Golay sequences have some properties that make it distinctive in the applications and results. However, for this distinction, the code sequences must be select carefully and accurately. The property of Golay complementary sequences can be expressed mathematically, where \( a_i \) and \( b_i \) \((i = 1, 2, \ldots, n)\) are the pair of binary complementary sequences of code length \(2^n\). The Aperiodic Auto-correlation Functions (AACFs) for the complementary sequences can be expressed as follows [1]:

\[
c_j = \sum_{i=1}^{n} a_i a_{i+j} \quad \ldots (1)
\]

and

\[
d_j = \sum_{i=1}^{n} b_i b_{i+j} \quad \ldots (2)
\]

The sum of the pair of AACFs can be expressed as;

\[
c_j + d_j = 0 \quad j \neq 0 \quad \text{for specific unique pair and there relates as in futures point 7}
\]

\[
c_j + d_j \neq 0 \quad j \neq 0 \quad \text{for other}
\]

\[
c_0 + d_0 = 2^m \quad \ldots (4)
\]

There are several recursive and one non-recursive methods for generating complementary sequences. The recursive method is used to generate the binary complementary sequences, which based on the following algorithm [1]:

\[
a_n = [a_{n-1}, b_{n-1}] \quad , \quad b_n = [a_{n-1}, -b_{n-1}] \ldots (5)
\]

Where the operator [ ] denotes concatenation of sequences, and \(a_n\) and \(b_n\) represent the complementary binary sequence of length \(2^n\). If DFT taken the above equation, then get [2].

\[
|c(k)|^2 + |d(k)|^2 = 2Nc \quad \ldots (6)
\]

There are many studies for generation a Golay complementary code sequences (GCCS). These are; Jimenez and etal 2002[3], built the GCCS simulation by Monte-Carlo algorithm, and then by DSP TMS320C6201. Borislav and etal 2004[4], present a computer algorithm to generate a GCCS for modern application. Carlos and Changho 2007[5,6], a recursive algorithm for reduce the computational of GCCS creation are present. Jonathan 2009 [7], generates GCCS from Barker code.

**PROPOSED CREATION ALGORITHM**

The efficient designs of a complementary set are depending on the generator (creator) and correlator complexity reduction. Therefore, to satisfy this requirement, there are some features must be identities, these are;
1. It has coupled complementary code sequences (A and B) with same length.
2. The output when use this code will get from eq’s (3) and (4).
3. This output does not produce from unique A and B, But maybe get from code sequences (A) with more than one code sequence (B) and vice versa as indicated in Table (1).
4. Therefore, the selection of A and B will be depended on the applications.
5. Not all formations of selected bits are suitable as Golay code sequences.
6. Other formations which can be suitable as Golay code sequences, which satisfy the eq’s (3) and (4) have the output as:
   a. Main loop =2N Side-loop = 0 as in figure(2-a section 1 and 2-b section 5).
   b. Main loop = 2N Side-loop ≠ 0 as in figure(2-a,b other sections).
7. The autocorrelation output of complementary codes A and B which gives sidelobe=0 can get from:
   \[ A \cdot B, -A \cdot B, A^* \cdot B, -A^* \cdot B, A \cdot B^*, -A \cdot B^*, A^* \cdot B^*, -A^* \cdot B^* \]
   Where:
   -A =inverse of A.
   A* = invert of A.
   -A* = invert (reverse (A)) = reverse (invert (A)).
   Also, its same for B.

The proposed algorithm are based on 8-bit as shown;
Let A and B a complementary Golay code sequences, and

\[
A = [A_7 \ A_6 \ A_5 \ A_4 \ A_3 \ A_2 \ A_1 \ A_0]; \\
-A = [-A_7 \ -A_6 \ -A_5 \ -A_4 \ -A_3 \ -A_2 \ -A_1 \ -A_0]; \\
A^* = [A_0 \ A_1 \ A_2 \ A_3 \ A_4 \ A_5 \ A_6 \ A_7]; \\
-A^* = [-A_0 \ -A_1 \ -A_2 \ -A_3 \ -A_4 \ -A_5 \ -A_6 \ -A_7];
\]

Also, for B are the same.
If \( X = A \ XOR (-A^*) \) \( \Rightarrow \) symmetry, and
\[ Y = B \ XOR (-B') \Rightarrow \text{symmetry.} \]
Then; for complementary A and B we will obtain;
\[
Y = \text{inverse (X)} \\
Y = -X = B \ XOR (-B')
\]
\[
Y_0 = B_0 \ XOR (-B_0') = B_0 \ XOR (-B_7) \\
-Y_0 = -(A_0 \ XOR (-A_0')) \Rightarrow (- (A_0 \ XOR (-A_7))); \text{ and}
Y_1 = B_1 \ XOR (-B_6) \\
Y_2 = B_2 \ XOR (-B_5) \\
Y_3 = B_3 \ XOR (-B_4)
\]

Then, we will get B codes, which represent the complementary of A from A, and to more reliable apply eq’s (3) and (4).

**SIMULATION AND IMPLEMENTATION RESULTS**
In Figure (1), flowchart and VHDL program to generate and correlate a Golay complementary code, which based on eight 8-bits implementation. Table (1) represents 192 Golay pair, which can be generating while only 48 can used because each four codes give the same response as explain in the features and as showed in the Table. We checked our algorithm to generate the Golay code for 4, 8, 10, and 16-bits, the result as in Table (2). Then, we simulate the correlation of 8-bit code after generate the code by proposed algorithm by matlab 2012 for check its work according to the equations (3) and (4), and the result as in Figure (2 and 3 ), after that, implement the algorithm by Xilinx-Spartan-3A XC3S700A FPGA, and the result as in Figure (4) which represents the A code with its related Bs. Figure(5) represents the autocorrelation output which related to simulation as in Figure(3-b). The hardware implementation as in Figure (6) by Xilinx-Spartan-3A XC3S700A FPGA.

The auto-correlation checking was made to the sidelobe, because it's equal to zero for the complementary pair code only, and for other, pair will be not equal zero. While for the all, the main will be equal to 2N as in eq. 3 and in Figure (2) and (3). In Figure(2-a) the complementary code takes place in section 1, while in Figure(2-b) it takes place in section 5. Figure(3-a) represents the whole autocorrelation adding process for (A’s) with all (B’s). While Figure(3-b) represents the output of adding process when sidelobe checking has been satisfied, where, only complementary code exists. Figure(5), represents the output of the same operation with Xilinx-Spartan-3A XC3S700A FPGA implementation as in Figure(6). It's clear from the figure the output only for a complementary pair according to sidelobe checking as in Figure(6) with green indicators. The correct output will be obtained when the complementary pairs are conformal.

CONCLUSIONS
The proposed implementation designs are based on 8-bits after simulate 4,8,10 and 16 bit as shown in the results. This algorithm takes less time than checking all possible numbers to find the required complementary code by 1/4 for 4-bit, 1/16 for 8-bit, 1/248 for 10 bit and 1/1024 for 16 bits. The implementation could modify for any number of bits as required the application need. This flexibility in the implementation was obtained from the use of Xilinx –Spartan-3 XC3S700 FPGA.

REFERENCES


```vhdl
a1:=A;
ad:=not a1;
for i in 0 to n1 loop
    x:=ad(i);ad(i):=ad(n-i);ad(n-i):=x;
end loop;

aa:=a1 xor ad;
bb:=not aa;
ind:=0;
for i in 0 to 15 loop
    x1(3 downto 0):=CONV_STD_LOGIC_VECTOR(i,4);
    for j in n1+1 to n loop
        x1(j):= bb(j) xor (not x1(n-j));
    end loop;
    xc:=conv(x1,n);
    ac:=conv(a1,n);
    cor(ac,xc,n,pp);
    zc:=0;
    for j in 0 to n loop
        if pp(j)=0 then
            zc:=zc+1;
        end if;
    end loop;
    if zc=n then
        ind:=ind+1;
        bx(ind):=x1;
    end if;
end loop;
B1<= bx(1);
B2<= bx(2);
B3<= bx(3);
B4<= bx(4);
```

Procedure `cor (x1,x2; in z;n:in integer; x3;out z) is` variable pa,pb,z;
for j in 0 to n loop
    pa(j):=0;
pb(j):=0;
for k in 0 to n+j loop
    pa(j):=pa(j)+x1(k)*x1(k+j);
pb(j):=pb(j)+x2(k)*x2(k+j);
end loop;
x3(j):=pa(j)+pb(j);
end loop;```

- **Figure (1) Proposed algorithm for generate and correlate GCCS.**
Implementation of Golay Complementary Code Sequences Generator Based on FPGA

Table (1) Golay Code Sequences formation (8 bit).

<table>
<thead>
<tr>
<th>A_{dec}</th>
<th>B_{dec}</th>
<th>A_{dec}</th>
<th>B_{dec}</th>
<th>A_{dec}</th>
<th>B_{dec}</th>
<th>A_{dec}</th>
<th>B_{dec}</th>
</tr>
</thead>
<tbody>
<tr>
<td>172</td>
<td>6</td>
<td>144</td>
<td>58</td>
<td>177</td>
<td>125</td>
<td>141</td>
<td>190</td>
</tr>
<tr>
<td>83</td>
<td>6</td>
<td>111</td>
<td>58</td>
<td>114</td>
<td>125</td>
<td>78</td>
<td>190</td>
</tr>
<tr>
<td>53</td>
<td>6</td>
<td>141</td>
<td>65</td>
<td>114</td>
<td>130</td>
<td>9</td>
<td>197</td>
</tr>
<tr>
<td>202</td>
<td>6</td>
<td>177</td>
<td>65</td>
<td>78</td>
<td>130</td>
<td>111</td>
<td>197</td>
</tr>
<tr>
<td>197</td>
<td>9</td>
<td>114</td>
<td>65</td>
<td>141</td>
<td>130</td>
<td>111</td>
<td>197</td>
</tr>
<tr>
<td>58</td>
<td>9</td>
<td>78</td>
<td>65</td>
<td>177</td>
<td>130</td>
<td>144</td>
<td>197</td>
</tr>
<tr>
<td>163</td>
<td>9</td>
<td>18</td>
<td>71</td>
<td>116</td>
<td>132</td>
<td>9</td>
<td>197</td>
</tr>
<tr>
<td>92</td>
<td>9</td>
<td>72</td>
<td>71</td>
<td>46</td>
<td>132</td>
<td>190</td>
<td>209</td>
</tr>
<tr>
<td>184</td>
<td>18</td>
<td>237</td>
<td>71</td>
<td>139</td>
<td>132</td>
<td>190</td>
<td>209</td>
</tr>
<tr>
<td>71</td>
<td>18</td>
<td>183</td>
<td>71</td>
<td>209</td>
<td>132</td>
<td>9</td>
<td>197</td>
</tr>
<tr>
<td>29</td>
<td>18</td>
<td>226</td>
<td>72</td>
<td>132</td>
<td>139</td>
<td>222</td>
<td>209</td>
</tr>
<tr>
<td>226</td>
<td>18</td>
<td>71</td>
<td>72</td>
<td>222</td>
<td>139</td>
<td>33</td>
<td>209</td>
</tr>
<tr>
<td>27</td>
<td>20</td>
<td>184</td>
<td>72</td>
<td>123</td>
<td>139</td>
<td>123</td>
<td>209</td>
</tr>
<tr>
<td>216</td>
<td>20</td>
<td>29</td>
<td>72</td>
<td>33</td>
<td>139</td>
<td>132</td>
<td>209</td>
</tr>
<tr>
<td>39</td>
<td>20</td>
<td>125</td>
<td>78</td>
<td>130</td>
<td>141</td>
<td>39</td>
<td>215</td>
</tr>
<tr>
<td>228</td>
<td>20</td>
<td>190</td>
<td>78</td>
<td>125</td>
<td>141</td>
<td>27</td>
<td>215</td>
</tr>
<tr>
<td>20</td>
<td>27</td>
<td>65</td>
<td>78</td>
<td>65</td>
<td>141</td>
<td>228</td>
<td>215</td>
</tr>
<tr>
<td>215</td>
<td>27</td>
<td>130</td>
<td>78</td>
<td>190</td>
<td>141</td>
<td>216</td>
<td>215</td>
</tr>
</tbody>
</table>

Figure (2) Output of equations (3,4) for complementary and not compatible complementary.
Implementation of Golay Complementary Code Sequences Generator Based on FPGA

Table (2) Golay complementary for different number of bits

<table>
<thead>
<tr>
<th>No. of bits</th>
<th>Total codes</th>
<th>Available codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>192</td>
<td>48</td>
</tr>
<tr>
<td>10</td>
<td>128</td>
<td>32</td>
</tr>
<tr>
<td>16</td>
<td>1536</td>
<td>384</td>
</tr>
</tbody>
</table>

Table (3) 8 bit Golay codes sequences (not repeated in A nor in B)

<table>
<thead>
<tr>
<th>A(Dec.)</th>
<th>B(Dec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>53</td>
</tr>
<tr>
<td>9</td>
<td>58</td>
</tr>
<tr>
<td>18</td>
<td>29</td>
</tr>
<tr>
<td>20</td>
<td>27</td>
</tr>
<tr>
<td>27</td>
<td>20</td>
</tr>
<tr>
<td>29</td>
<td>18</td>
</tr>
<tr>
<td>33</td>
<td>46</td>
</tr>
<tr>
<td>46</td>
<td>33</td>
</tr>
<tr>
<td>53</td>
<td>6</td>
</tr>
<tr>
<td>58</td>
<td>9</td>
</tr>
<tr>
<td>65</td>
<td>78</td>
</tr>
<tr>
<td>78</td>
<td>65</td>
</tr>
</tbody>
</table>
Implementation of Golay Complementary Code Sequences Generator Based on FPGA

Figure (3). Output of equations (3,4): a) all output for each correlation process b) only sidelobe=0 output.

Figure (4) generate complementary B for A and from A.
Figure (5) Output of the correlation process as built by FPGA.

Figure (6) Implementation Circuit for Generate and Correlate of GCCS.