Technical Report

Golay and Wavelet Error Control Codes in VLSI

Arunkumar Balasundaram, Angelo Pereira, Jun Cheol Park and Vincent Mooney

Georgia Institute of Technology, Atlanta, Georgia, U.S.A

December 2003

1. INTRODUCTION

This technical report describes AGNI (meaning *Fire* in Sanskrit) – a VLSI chip to implement error control codes. The chip was initially conceived and designed as part of a Georgia Tech Cutting Edge Research Grant. However, this chip implementation of error control codes has been undertaken as a part of the ECE 6130 course taught in Spring 2002 by Dr. John Uyemura, Professor, Department of Electrical and Computer Engineering, Georgia Institute of Technology. Two coders have been implemented: a (12, 6, 4) wavelet encoder/decoder and a (24, 12, 8) golay encoder/decoder, where the (N, M, d) nomenclature stands for (N=encoded length, M=message length, d=hamming distance). These codes have a correctable limit of one bit error and three bit errors, respectively. The following section presents the encoding/decoding functionality of the chip in more detail.

This project could potentially feed a future project to incorporate the chip into a System-on-a-Package (SoP). It is expected that the chip would function as a high-speed error encoder/decoder for Radio Frequency (RF) applications.

2. ENCODING/DECODING FUNCTIONALITY OF THE CHIP

Table I presents examples of wavelet and wavelet-based golay encoding schemes [1], in which we put a comma after every 4 digits to enhance readability. The decoding of wavelet and golay codes with one bit error and three bit errors, respectively, have been shown in Table II.

TABLE I
Examples of wavelet and golay encoding schemes

	Message	Code
Wavelet	00,0001	0100,1101,0011
Golay	0000,0000,0001	1010,0011,0011,1100,0001,0011

TABLE II
Examples of wavelet and golay decoding schemes
with bit errors

	code	Message
Wavelet	0100,1100,0011	00,0001
Golay	1010,0011,0111,0101,0001,0011	0000,0000,0001

The design achieves the encoding/decoding functionality in combinational logic. It has been found that all blocks, except the golay decoder, can be simplified to a single stage combinational logic block. This simplification has helped us achieve higher clock speeds of approximately 150 MHz and data transfer rates of the order of 900 Megabit/sec using a 0.25 μm CMOS library provided by Artisan, Inc. [2].

3. DESIGN APPROACH

The design has been achieved using a typical design flow for designing an Integrated Circuit (IC). Figure 1 illustrates this design flow and the tools that have been used to implement the various steps. The following subsections present more details of each of these steps.

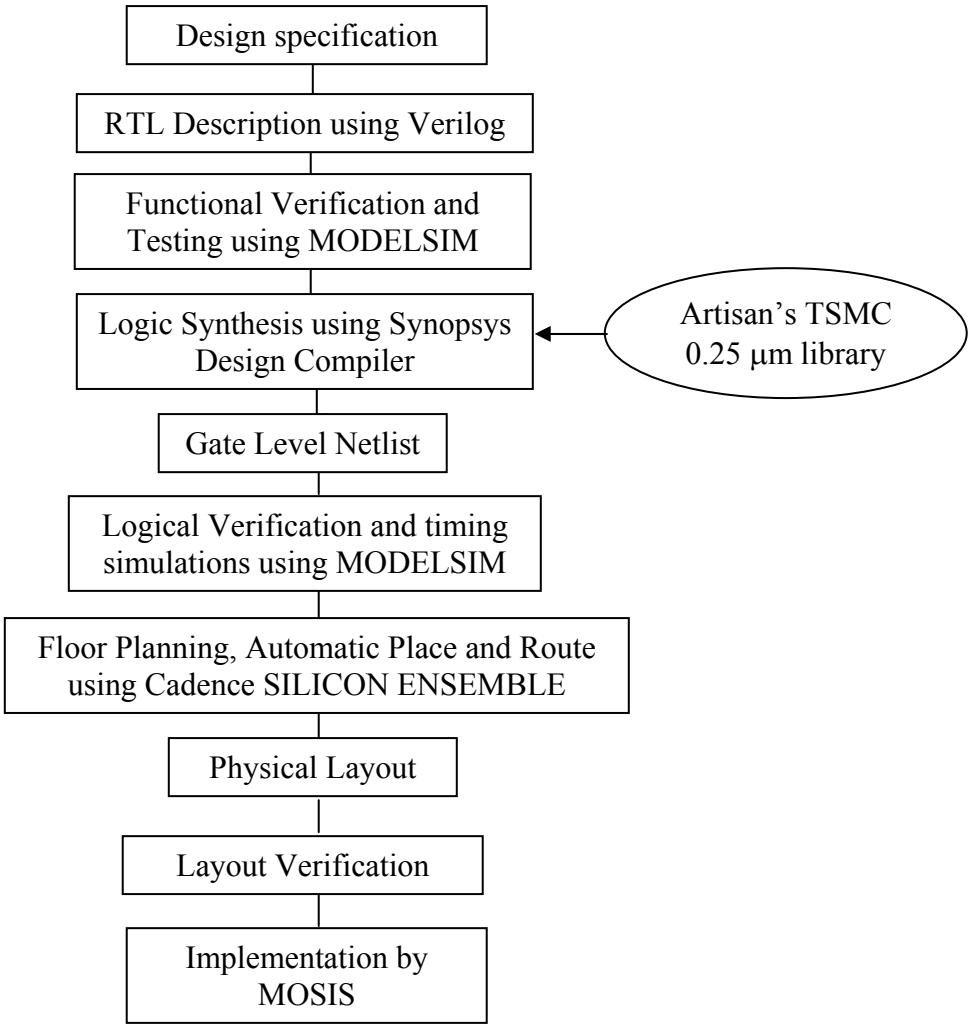

Figure 1: Design Flow of the circuit

3.1. Register Transfer Level (RTL) description using Verilog

The RTL description of the circuit has been done using Verilog and the corresponding data flow is shown in Figure 2. The design is fully synchronous and positive edge triggered.

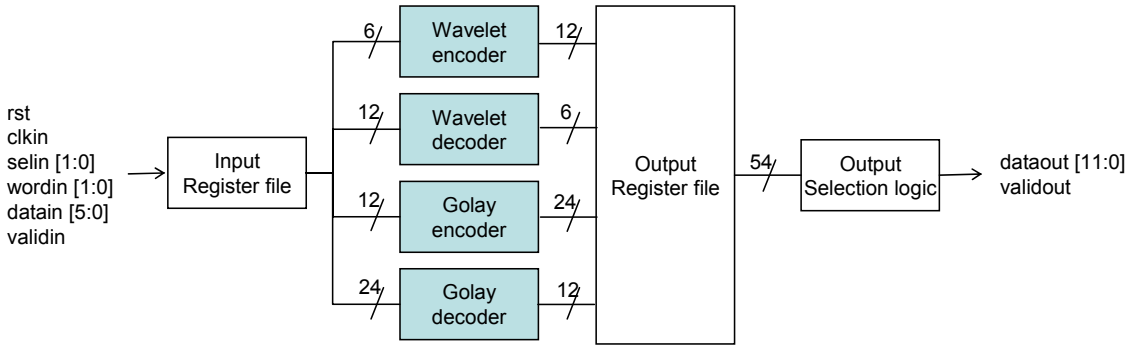

Figure 2: Data flow description of the circuit

The core of the design consists of four modules – wavelet encoder, wavelet decoder, golay encoder and golay decoder. The input/output specifications of the four modules are shown in Table III. The first three modules, i.e., wavelet encoder, wavelet decoder and golay encoder are single stage combinational blocks and take one cycle to compute the output. The last module, the golay decoder, is a multi-stage combinational block and takes 12 cycles to compute its output.

TABLE III
Input/output specifications of the encoder/decoder modules

Module	Input size (bits)	Output size (bits)
Wavelet encoder	6	12
Wavelet decoder	12	6
Golay encoder	12	24
Golay decoder	24	12

The total number of inputs and outputs in Figure 2 is 26. The number of data input bits (six) and output bits (twelve) is chosen to optimize the wavelet encoder case; another motivation in choosing only 26 inputs and outputs is to keep packing costs low.

3.1.1. Wavelet encoder

The four encoder and decoder modules shown as shaded blocks in Figure 2 are implemented based on the algorithm structure explained in [9]. Since this report focuses on the hardware implementation of the four encoders and decoders, the detailed algorithmic derivations are not fully described. Interested readers may refer to [9] for an in-depth explanation of the algorithms used.

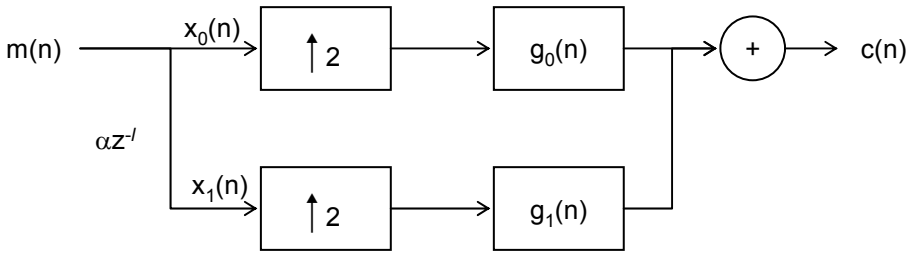

Figure 3: Filter bank structure of the wavelet encoder

Figure 3 shows the encoder of an (N, M, d) wavelet code that maps the message block m(n) of size M = N/2 to the codeword c(n) where n indicates the nth sequence (0 < n < N-1). 'd' is the hamming distance. The wavelet encoder uses (N=12, M=6, d=4). In Figure 3, x0(n) and x1(n) are wavelet coefficients. The synthesis filters $g_0(n)$ and $g_1(n)$ are the scaling sequence and mother wavelet, respectively. In [9], the relationship and values for the wavelet encoder of synthesis filters are given as follows.

$$g_0((n))_N = g_1((-n-1))_N \quad (1)$$

$$g_0(n) = \{101010010001\},\ g_1(n) = \{100010010101\} \quad (2)$$

where $((\cdot))_N$ denotes a modulo-N operation. The filter bank structure of the wavelet encoder shown in Figure 3 is expressed as follows:

$$c(n) = (G_0 x_0)(n) + (G_1 x_1)(n), \quad (3)$$

in which *G0* and *G1* are 2-circulant matrices (in an n-circulant matrix, each row is identical to the previous row but n-position shifted and wrapped around). The upsampling of periodic signals by a factor two followed by a filtering operation can be represented by the algebra of 2-circulant matrices. The 2-circulant matrix *Gj* can be derived from following equations.

$$Gj = H_j^T \tag{4}$$

$$H_j = \begin{bmatrix} g_j(0) & g_j(1) & g_j(2) & \cdots & g_j(N-1) \\ g_j(N-2) & g_j(N-1) & g_j(0) & \cdots & g_j(N-3) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ g_j(2) & g_j(3) & g_j(4) & \cdots & g_j(1) \end{bmatrix} \quad j = 0, 1. \tag{5}$$

In Equation 3, $x_1(n)$ is $\alpha z^{-l} \cdot m(n)$ (i.e., $\alpha z^{-l} \cdot x_0(n)$), in which the product of the parameter α and the delay z^{-l} can be expressed as $\prod_l$. $\prod_l$ is a one-circulant matrix, in which each element of the first row is zero except for the (l+1) position. The delay l can be chosen any value from the set {1, 2, 4, 5} to satisfy the required hamming distance of four. It may be noted that codes generated by different choices of l are all equivalent. We use l=5 for the wavelet encoder and decoder. As a result, Equation 3 can be expressed as follows.

$$c(n) = (G_0 + G_1 \prod_l)(x_0(n)) \tag{6}$$

Because we use one-bit operation, operation $a + b$ is equivalent to a *XOR* b (i.e., $a \oplus b$). Finally, the codeword c(n) is expressed with message block m(n) as in Equation 7.

$$\begin{aligned} c(0) &= m(0) \oplus m(3) \oplus m(4) \\ c(1) &= m(0) \oplus m(2) \oplus m(3) \\ c(2) &= m(1) \oplus m(4) \oplus m(5) \\ c(3) &= m(1) \oplus m(3) \oplus m(4) \\ c(4) &= m(0) \oplus m(2) \oplus m(5) \\ c(5) &= m(2) \oplus m(4) \oplus m(5) \\ c(6) &= m(0) \oplus m(1) \oplus m(3) \\ c(7) &= m(0) \oplus m(3) \oplus m(5) \\ c(8) &= m(1) \oplus m(2) \oplus m(4) \\ c(9) &= m(0) \oplus m(1) \oplus m(4) \\ c(10) &= m(2) \oplus m(3) \oplus m(5) \\ c(11) &= m(1) \oplus m(2) \oplus m(5) \end{aligned} \tag{7}$$

Figure 4 shows the logic diagram which implements the expressions in Equation 7.

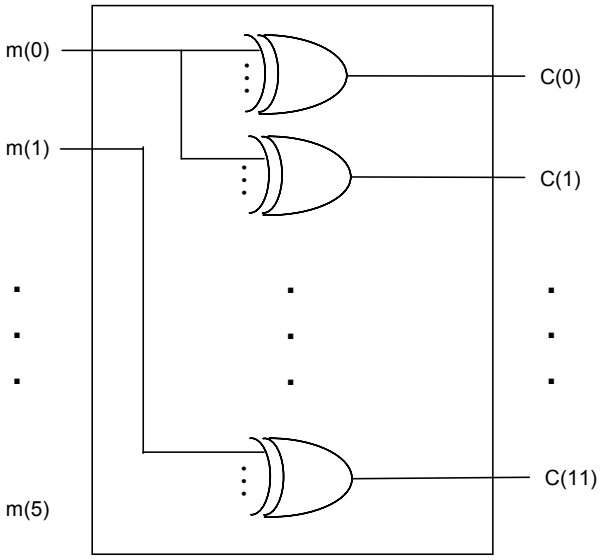

Figure 4: Logic diagram of the wavelet encoder

3.1.2. Wavelet decoder

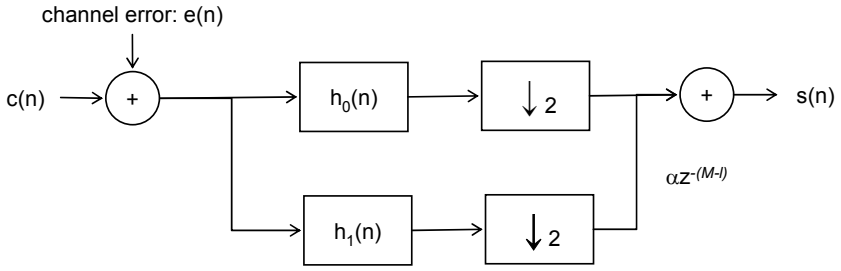

Figure 5: Filter bank of the syndrome generator

The syndrome of the wavelet is shown in Figure 5, in which s(n) is the syndrome and e(n) is an error pattern due to the communication channel. A syndrome is a sequence that is uniquely associated with a set of codewords within hamming distance d (note that the wavelet code nomenclature (N, M, d) stands for the encoded length, message length and hamming distance as explained in Section 1). Therefore, if the number of error bits occurring during data transmission is equal to or less than (d-1), then we can detect that the received codeword is not valid. However, the number of correctable error bits is smaller than the number of detectible error bits. $h_0(n)$ and $h_1(n)$ are two analysis filters that are given as follows:

$$h_j((n))_N = h_j((-n))_N \quad j=0,1\ n=0,\ldots,N-1 \tag{8}$$

$$h_0(n) = \{110001001010\},\ h_1(n) = \{110101001000\}. \tag{9}$$

The filter bank shown in Figure 5 can be expressed as follows:

$$s(n) = (H_0 + H_1 \prod_l^T)(e(n)) = H_2(e(n)) \tag{10}$$

where H_2 is 2-circulant matrix of h_2, i.e., $h_0(n) + h_1((n+2l))_N$. h_2 is given as follows:

$$h_2(n) = \{111100011000\} \tag{11}$$

where H_2 is constructed using the relation $h_2((2n-i))_N$. To reconstruct m(n), we interpolate the low (M) dimensional syndrome s(n) into the higher (N) dimensional error pattern e(n). The given decoder algorithm in [9] inverts the polyphase filters of the syndrome generator filter $h_2(n)$. The two polyphase filters of $h_2(n)$ are as follows:

$$\begin{aligned} u_{00}(n) &= h_2(2n) = \{110010\}, \\ u_{01}(n) &= h_2(2n+1) = \{110100\}. \end{aligned} \tag{12}$$

The two inverting filters $r_{00}(n)$ and $r_{01}(n)$, which are the circulat inverses of $u_{00}(n)$ and $u_{01}(n)$, respectively, are defined as follows:

$$\begin{aligned} &R_{0i}(z)\, U_{0i}(z) = 1 \bmod (z^{-M} - 1) \qquad i=0,1 \\ &r_{00}(n) = \{001011\}, \\ &r_{01}(n) = \{011010\} \end{aligned} \tag{13}$$

$$R_{00}*U_{00} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \quad R_{01}*U_{01} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

In Equation 13, R_{0i} and U_{0i} are one-circulant matrices. Note the $R_{01}*U_{01}$ matrix shows z^{-1} delay from the $R_{00}*U_{00}$ matrix. Then, the reconstruction algorithm is given as Figure 6.

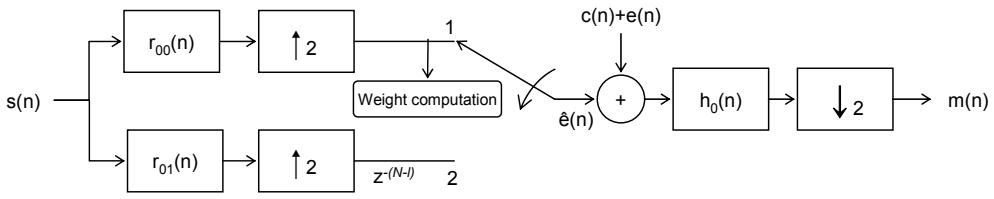

Figure 6: Filter structure to reconstruct the message sequence

The algorithm generates the message m(n) from syndrome s(n). The algorithm selects node1 or node2 according to the weight computation, in which node 2 is selected whenever the weight is 5. The error weight wt(e) is given by $R_{00}*H_{2.}$

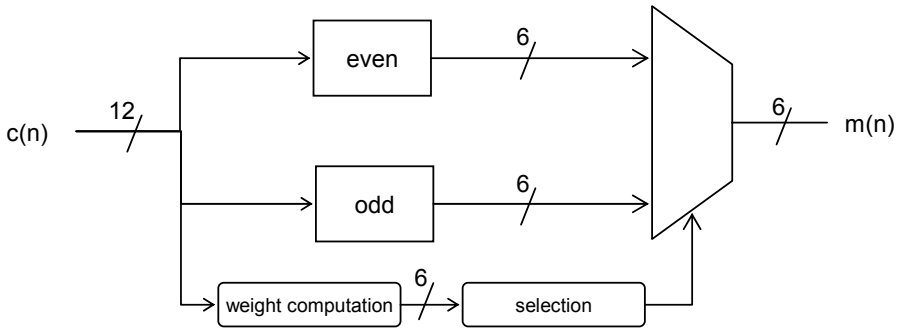

Figure 7: Logic diagram of the wavelet decoder architecture

even

m(0) = c(5) ⊕ c(9) ⊕ c(11)
m(1) = c(1) ⊕ c(7) ⊕ c(11)
m(2) = c(1) ⊕ c(3) ⊕ c(9)
m(3) = c(3) ⊕ c(5) ⊕ c(11)
m(4) = c(1) ⊕ c(5) ⊕ c(7)
m(5) = c(3) ⊕ c(7) ⊕ c(9)

odd

m(0) = c(2) ⊕ c(4) ⊕ c(8)
m(1) = c(4) ⊕ c(6) ⊕ c(10)
m(2) = c(0) ⊕ c(6) ⊕ c(8)
m(3) = c(2) ⊕ c(8) ⊕ c(10)
m(4) = c(0) ⊕ c(4) ⊕ c(10)
m(5) = c(0) ⊕ c(2) ⊕ c(6)

weight computation

wt(0) = c(0) ⊕ c(1) ⊕ c(3) ⊕ c(5) ⊕ c(7) ⊕ c(9)
wt(1) = c(2) ⊕ c(3) ⊕ c(5) ⊕ c(7) ⊕ c(9) ⊕ c(11)
wt(2) = c(1) ⊕ c(4) ⊕ c(5) ⊕ c(7) ⊕ c(9) ⊕ c(11)
wt(3) = c(1) ⊕ c(3) ⊕ c(6) ⊕ c(7) ⊕ c(9) ⊕ c(11)
wt(4) = c(1) ⊕ c(3) ⊕ c(5) ⊕ c(8) ⊕ c(9) ⊕ c(11)
wt(5) = c(1) ⊕ c(3) ⊕ c(5) ⊕ c(7) ⊕ c(10) ⊕ c(11)

(14)

We implement the wavelet decoding architecture by optimizing the algorithm given in Figures 5 and 6. The resulting optimized architecture (and algorithm) is as shown in

Figure 7. The logic blocks "even" and "odd" generate message m(n) from codeword c(n) using Equation 14. The even and odd logic blocks can be derived from the following expressions:

$$\text{even} = H_0*(R_{00*}H_2+I)*c(n)$$
$$\text{odd} = H_0*(z^{-11}*R_{01*}H_2+I)*c(n). \quad (15)$$

The weight computation logic computes weights using Equation 14. The selection logic takes the output of the "odd" logic block as a message m(n) when the result of the weight computation has 5 ones and 1 zero, i.e., weight=5.

3.1.3. Wavelet-base golay encoder

The wavelet-based golay encoder uses a similar algorithm to the wavelet encoder but has different sizes for the message block and codeword (i.e., (N=24, M=12, d=8)). Consequently, the golay encoder uses 24-bit filters instead of the 12-bit filters used in Equation 2. The scaling sequence and mother wavelet for golay are given as follows:

$$g_0(n) = \{A80011\},\ g_1(n) = \{40DD55\}. \quad (16)$$

In the results in [9], the golay encoder does not use the delay element. Hence equation 6 is modified for the golay encoder as follows:

$$c(n) = (G_0 + G_1)(x_0(n)). \quad (17)$$

The 2-circulant matrices $G_0 + G_1$ can be derived from Equation 4 and 5. As a result, the codeword c(n) is expressed as follows:

$$\begin{aligned}
c(0) &= m(0) \oplus m(6) \oplus m(8) \oplus m(10) \oplus m(11)\\
c(1) &= m(0) \oplus m(2) \oplus m(4) \oplus m(5) \oplus m(6) \oplus m(7) \oplus m(8)\\
c(2) &= m(0) \oplus m(1) \oplus m(7) \oplus m(9) \oplus m(11)\\
c(3) &= m(1) \oplus m(3) \oplus m(5) \oplus m(6) \oplus m(7) \oplus m(8) \oplus m(9)\\
c(4) &= m(0) \oplus m(1) \oplus m(2) \oplus m(8) \oplus m(10)\\
c(5) &= m(2) \oplus m(4) \oplus m(6) \oplus m(7) \oplus m(8) \oplus m(9) \oplus m(10)\\
c(6) &= m(1) \oplus m(2) \oplus m(3) \oplus m(9) \oplus m(11)\\
c(7) &= m(3) \oplus m(5) \oplus m(7) \oplus m(8) \oplus m(9) \oplus m(10) \oplus m(11)\\
c(8) &= m(0) \oplus m(2) \oplus m(3) \oplus m(4) \oplus m(10)\\
c(9) &= m(0) \oplus m(4) \oplus m(6) \oplus m(8) \oplus m(9) \oplus m(10) \oplus m(11)\\
c(10) &= m(1) \oplus m(3) \oplus m(4) \oplus m(5) \oplus m(11)\\
c(11) &= m(0) \oplus m(1) \oplus m(5) \oplus m(7) \oplus m(9) \oplus m(10) \oplus m(11)\\
c(12) &= m(0) \oplus m(2) \oplus m(4) \oplus m(5) \oplus m(6)\\
c(13) &= m(0) \oplus m(1) \oplus m(2) \oplus m(6) \oplus m(8) \oplus m(10) \oplus m(11)\\
c(14) &= m(1) \oplus m(3) \oplus m(5) \oplus m(6) \oplus m(7)\\
c(15) &= m(0) \oplus m(1) \oplus m(2) \oplus m(3) \oplus m(7) \oplus m(9) \oplus m(11)\\
c(16) &= m(2) \oplus m(4) \oplus m(6) \oplus m(7) \oplus m(8)\\
c(17) &= m(0) \oplus m(1) \oplus m(2) \oplus m(3) \oplus m(4) \oplus m(8) \oplus m(10)\\
c(18) &= m(3) \oplus m(5) \oplus m(7) \oplus m(8) \oplus m(9)\\
c(19) &= m(1) \oplus m(2) \oplus m(3) \oplus m(4) \oplus m(5) \oplus m(9) \oplus m(11)\\
c(20) &= m(4) \oplus m(6) \oplus m(8) \oplus m(9) \oplus m(10)\\
c(21) &= m(0) \oplus m(2) \oplus m(3) \oplus m(4) \oplus m(5) \oplus m(6) \oplus m(10)\\
c(22) &= m(5) \oplus m(7) \oplus m(9) \oplus m(10) \oplus m(11)\\
c(23) &= m(1) \oplus m(3) \oplus m(4) \oplus m(5) \oplus m(6) \oplus m(7) \oplus m(11)
\end{aligned} \quad (18)$$

Figure 8 shows the logic diagram, which implements the expressions in Equation 18.

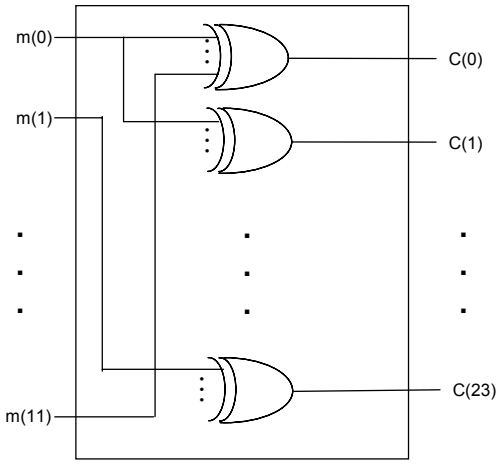

Figure 8: Logic diagram of the golay encoder

3.1.4. Golay decoder

For the wavelet-based golay decoder, filters are given in [10] as follows:

$$g_0(n) = \{A80011\},\ g_1(n) = \{40DD55\},$$
$$h_2(n) = \{915D8B\}, \qquad (19)$$
$$r_{00}(n) = \{100101001001\},\ r_{01}(n) = \{101100110110\}.$$

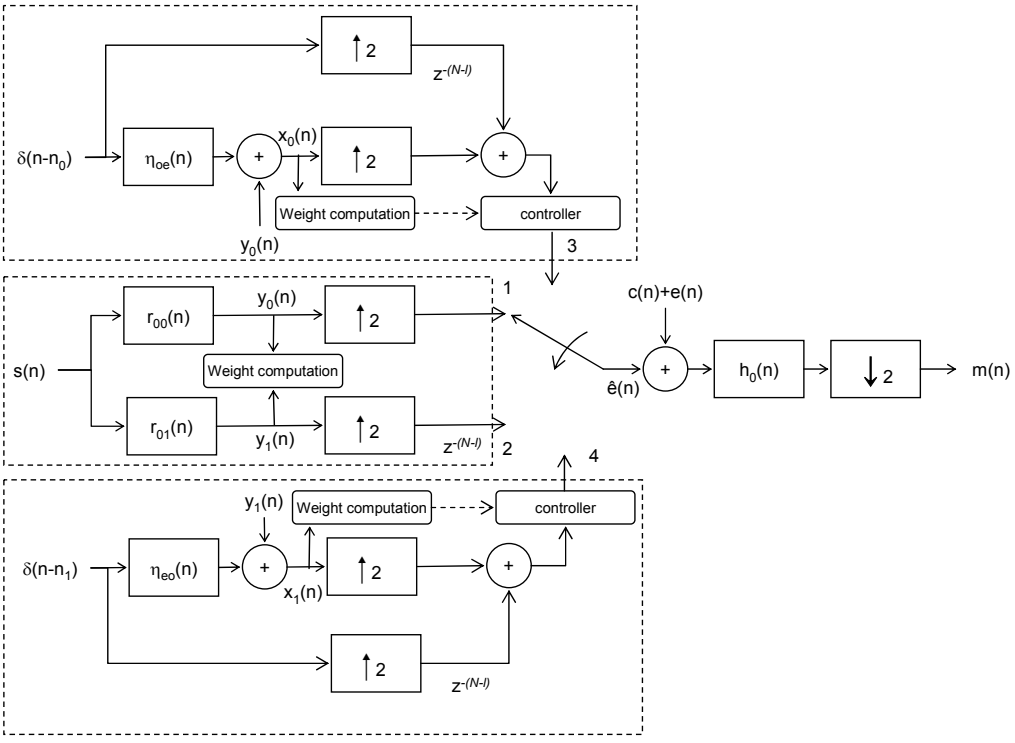

Figure 9: Wavelet-based golay decoder to reconstruct message m(n)

Figure 9 shows how the golay decoder algorithm generates message m(n), in which ηεo and ηoε are defined as follows:

$$\eta\varepsilon o = u_{01}(n) * r_{00}(n),\ \eta o\varepsilon = u_{00}(n) * r_{01}(n). \quad (20)$$

Similar to the wavelet decoder, a golay decoder generates an appropriate error patter e(n) according to the error weight. The error weight $wt(y_0)$ and $wt(y_1)$ can be obtained in a manner similar to the wavelet decoding scheme. Since the weights $wt(x_0)$ and $wt(x_1)$ are dependent on n_0 and n_1, respectively, the weights for the x_0 and x_1 sequences need to be calculated at most M times, i.e., 12 times, to obtain desired error weight corresponding to the most likely bit errors while varying n_0 and n_1.

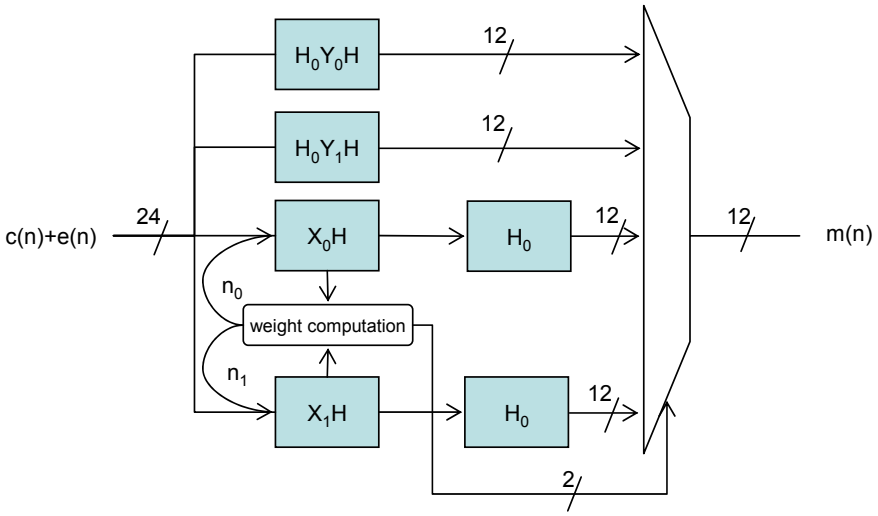

Figure 10: Logic diagram of the wavelet-based golay decoder

Figure 10 shows the logic diagram of the wavelet-based golay decoder. The logic blocks H_0Y_0 and H_0Y_1 are typical syndrome generators. Logic blocks X_0 and X_1 calculate the weight while varying n_0 and n_1. Unlike the algorithm in [10], which processes X_0 and X_1 weight computation in serial and thus takes up to 24 iterations, we use parallel weight computation. The use of parallel weight computation results in a maximum required number of iterations of 12. The calculated weight from the weight computation logic decides appropriate message sequences.

3.1.5. Inputs to the circuit

We now describe input/output pins and functional blocks other than shaded encoder/decoder modules in Figure 2.

(a) rst: rst is an asynchronous reset, for all internal flip-flops. The positive edge is used.

(b) clkin: *clkin* is the pin providing the input clock to all the flip-flops. The positive edge of clkin is used to update the outputs.

(c) selin [1:0]: *selin* is a two bit wide signal to select among the four modules available. At any given time only one of the encoders/decoder modules may be in use. Table IV lists the select signal for the four modules.

TABLE IV
Select signal for the encoder/decoder modules

Module	selin [1:0]
Wavelet encoder	00
Wavelet decoder	01
Golay encoder	10
Golay decoder	11

(d) wordin [1:0]: *wordin* is a 2-bit wide signal that indicates which segment of a multiple word sequence (dictated by the module that is chosen by *selin*) is given in a clock cycle. For example, if the multi-word input to the golay decoder is 0xa37513 [1010 0011 0111 0101 0001 0011], the sequence of the words, given as datain [5:0], in the various clock cycles is shown in Table V.

TABLE V
Sequencing of multi - word input

Cycle	wordin [1:0]	datain [5:0]
0	11	101000
1	10	110111
2	01	010100
3	00	010011

(e) datain [5:0]: *datain* is a 6-bit wide input bus, that brings in a single binary word for encoding or decoding. In conjunction with the wordin [1:0] signal, it is used to input multiple word sequences to the circuit.

(f) validin: *validin* is a signal used to indicate start/stop/resume of data transmission. A value of '1' on this line indicates that the current values of datain [5:0] are valid. A value of '0' indicates the converse.

3.1.6. Outputs of the circuit

(a) dataout [11:0]: *dataout* is the 12-bit wide output bus of AGNI.

(b) validout: *validout* is the complementary signal to *validin* on the output side. The *dataout* is valid only when *validout* signal is '1'.

3.1.7. Input register file

This module has a 24-bit register with four 6-bit segments, which receive and store the input data. If the four 6-bit segments are labeled as [5:0], [11:6], [17:12], [23:18], Table VI shows the data flow from the input register file to the encoder/decoder modules in conjunction with the selin [1:0] and wordin [1:0] signals in the various clock cycles.

TABLE VI
Data flow from the Input register file to the encoder/decoder modules

Modules	Segments of input register file activated
Wavelet encoder	[5:0]
Wavelet decoder	[5:0],[11:6]
Golay encoder	[5:0],[11:6]
Golay decoder	[5:0],[11:6],[17:12],[23:18]

3.1.8. Output register file

The 54-bit output register file in Figure 2 consists of four registers, one for each encoder/decoder module. The register widths correspond to the size of the output produced by the respective encoder/decoder module.

3.1.9. Output selection logic

The output selection logic selects 12-bit data from the 54-bit output register file corresponding to one output of the four encoder/decoder modules as shown in Figure 2. In three cases, the 12-bit data contains the entire output; in the fourth case, the wavelet-based golay encoder case, the output consists of 24 bits. Thus, for the wavelet-based golay encoder output, the 24-bit data output is communicated in two consecutive clock cycles.

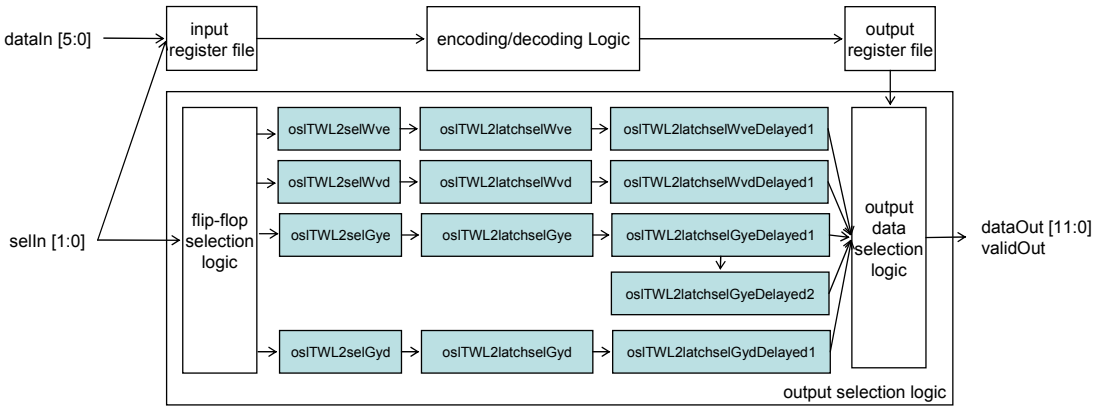

Figure 11: Logic diagram including internal details of the output selection logic

Figure 11 shows the internal structure of the output selection logic. The *selIn* signal in Figure 11 is connected to flip-flop selection logic in the output selection logic. The output selection logic has four internal pipelines, consisting of the edge-triggered flip-flops shown as shaded boxes in Figure 11. The flop-flop selection logic transfers a "1" to one of the four flip-flops connected to the flip-flop selection logic according to the *selIn* signal. Each flip-flop in the internal pipelines transfers its input (which, in the case of a "1" input, is a control signal) to the next flip-flop in each clock. The pipeline for the golay encoder has one extra stage since the golay encoder requires two cycles to transfer a 24-bit output to the 12-bit wide *dataOut* port. The output data selection logic selects 12-bit data from the output register file according to the control signals from the flip-flops of the internal pipelines. The output data selection logic transfers the selected 12-bit data to the *dataOut* port; the output data selection logic also generates the *validOut* signal for the *validOut* port.

3.2. Functional verification and testing using MODELSIM

The following tests are used in the verification effort.

3.2.1. Basic tests for each module

Four tests have been written, one for each encoder/decoder module. The tests for the encoder are exhaustive, i.e., all message words have been used and thus all corresponding code words have been generated. For the decoders, both clean code words as well as some subset of randomly corrupted code words have been used. We verified that the decoders are able to correct errors within their respective correctable limit. For the wavelet decoder, the correctable limit is one bit, while for the wavelet-based golay decoder the correctable limit is three bits. Since we initially targeted 50 Mhz as a worst case operational clock frequency, all testbenches use a 20 ns clock (later, we were able to reach almost 190 Mhz). The test vector set and expected result set for each of these tests have been generated from reference designs implemented in MATLAB.

The simulations have been performed using Mentor Graphics' MODELSIM simulator [3]. Simulation results are available as a MODELSIM waveform, which are shown as Figures 12, 13, 14 and 15 for the wavelet encoder, wavelet decoder, golay encoder and golay decoder, respectively. The simulation diagrams show the progression of data flow from data input to clocking into the input register file, the code or message generated by the appropriate encoder/decoder module and it's clocking into the output register file to finally select the appropriate module for output to be released to the external world. In Figures 12, 13, 14 and 15, the signal *irfTBGdataOut1* refers to the input register file. The signals *wveTBG2code* (Figure 12) and *gyeTBG2code* (Figure 13) refer to the codes generated by the wavelet encoder and golay encoder, respectively. The signals *wvdTBG2mesg* (Figure 14) and *gydTBG2mesg* (Figure 15) refer to the messages generated by the wavelet decoder and golay decoder, respectively. The signals *orfTBG2wveOutReg* (Figure 12), *orfTBG2wvdOutReg* (Figure 13), *orfTBG2gyeOutReg* (Figure 14) and *orfTBG2gydOutReg* (Figure 15) refer to the appropriate segments of the output register file corresponding to wavelet encoder, wavelet decoder, golay encoder and golay decoder, respectively.

In Figure 12, the signal *selIn*=0 means that the wavelet encoder is active. The second input value in *dataIn* is 0x01 after the *validIn*=1. The wavelet encoder generates corresponding output 0x2d3 through the *dataOut* port 3 cycles after the data input

The wavelet decoder simulation is shown in Figure 13 (*selIn*=1). To demonstrate the error correction functionality of the wavelet decoder, we took the output data from the *dataOut* in Figure 12 and modified 1-bit of 12 bits. Then the modified 12-bit is used as data inputs for the *dataIn* in Figure 12. The wavelet decoder requires a 12-bit data input and the data input port is 6-bit wide. Therefore, the 12-bit input data are inserted in 2 cycles in a serial manner. The most significant 6 bits are inserted first with the signal *wordIn*=1 and then the least significant 6 bits are inserted with the signal *wordIn*=0. In Figure 13, at the third and the fourth cycles after *valid*=1, the *dataIn* inputs are 0x0b and 0x03, respectively. These inputs represent binary values 001011 and 000011, respectively, which, when placed together, form {001011, 000011} (=0x2c3). Please note the combined number has a one bit error from {001011, 010011} (=0x2d3), which is the encoded version of 0x01. The wavelet decoder generates an error corrected output signal 000001 (=0x01) using the 6-bit *dataOut* port. The *dataOut* output is valid as indicated by the *validOut* signal 3 cycles after the input of the least significant 6 bits of the encoded bit pattern input to the wavelet decoder.

Figure 14 shows a golay encoder simulation (*selIn*=2). The golay encoder also needs 2 cycles to accept 12-bit input data. Similar to the wavelet decoder, the most significant 6 bits are inserted first with the signal *wordIn*=1 and then the least significant 6 bits are inserted with the signal *wordIn*=0. In Figure 14, at the third and fourth cycles after *validIn*=1, the *dataIn* inputs are 0x00 and 0x01, respectively. The combination of the two inputs forms 0x001. Three cycles after the input of the least significant 6 bits of the undecoded 12-bit input pattern, the golay encoder module returns corresponding encoded outputs 0xa37 and 0x513 through the *dataOut* port. Since the output of the golay encoder is 24 bits wide and the output port of the golay encoder is 12 bits wide, the output of the golay encoder is returned in two cycles, in which the most significant 12 bits are returned first and then the least significant 12 bits are returned next.

Figure 15 shows golay decoder simulation (*selIn*=3). To demonstrate error correction functionality of the golay decoder, we took data outputs from the *dataOut* signal of the golay encoder in Figure 14 and modified one of the 24 bits to introduce 1-bit errors. Then the modified 24-bit data are used as data inputs to the *dataIn* port in Figure 15. Therefore, the data output from the *dataOut* signal in Figure 15 should be golay decoded to the

corresponding data input from in Figure 14 (if the golay decoder error correction works correctly). For example, output data {0xa37, 0x513} from the *dataOut* signal in Figure 14 are modified in one bit location and inserted as a data input for the *dataIn* signal in Figure 15 (please note that the data 0xe37513 in the *irfTBG2dataOut1* in Figure 15 have one bit different from {0xa37, 0x513}). The golay decoder input is 24 bits wide, while the data input port *dataIn* is 6 bits wide. Therefore, 4 cycles are required to insert 24-bit input data. When the signal *validIn*=1, [23:18] of the input data is inserted with the signal *wordIn*=3; [17:12] of the input data is inserted with the signal *wordIn*=2; [11:6] of the input data is inserted with the signal *wordIn*=1; and [5:0] of the input data is inserted with the signal *wordIn*=0. In Figure 15, the inserted input data sets {0x38, 0x37, 0x14, 0x13}, which are {111000, 110111, 010100, 010011} in binary numbers, are the one bit error representation of {0x28, 0x37, 0x14, 0x13} (=101000, 110111, 010100, 010011). These numbers can be represented as {0xa37, 0x513} (=101000110111, 010100010011), which are the encoded version of {0x00, 0x01} (=000000, 000001). In this example, the golay decoder module returns corresponding output 0x001 through the *dataOut* port after 15 cycles of the valid *dataIn*[5:0] input, correcting one bit error. The golay decoder can correct errors of up to 3 bits. As mentioned earlier in Section 3.1, the wavelet encoder/decoder and the golay encoder take one cycle to compute the output while the golay decoder takes 12 cycles. The extra 3 cycles (15-12=3) are required to prepare input and output data.

3.2.2. Tests for dynamic switching between any two modules

A single testbench has been written to test dynamic switching from one module to another. Figure 16 shows a dynamic transition between wavelet encoder and wavelet decoder without losing a single cycle. In Figure 16, the first *tbTWL2validIn* signal, which represents the *validIn* signal in the encoder/decoder modules, lasts for 4 cycles. During the 4 cycles, the *tbTLB2selIn* signal, which represents the *selIn* signal in the encoder/decoder modules, changes from a “0” to a “1” and then changes back to a “0.” This means that the testbench first activates the wavelet encoder, second activates the wavelet decoder and third activates the wavelet encoder again. The testbench takes a data input 0x01 using the *tbTBL2dataIn* signal and generates an encoded number 0x2d3 to the *tbTBL2dataOut* outputs. Then the testbench takes two data input {0x0b, 0x13} and generates a decoded number 0x001. Finally, the testbench takes a data input 0x33 and generates an encoded number 0x606.

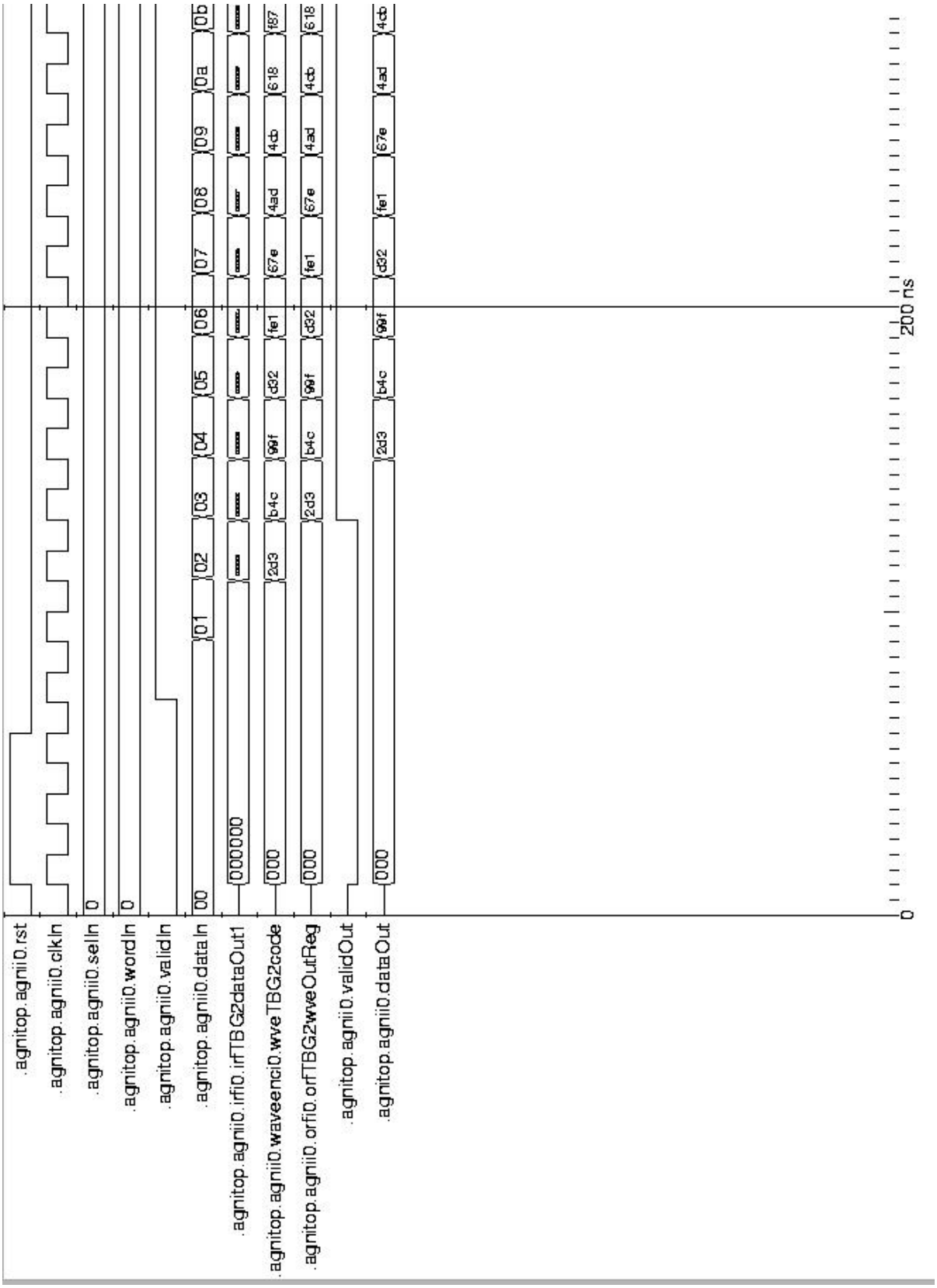

Figure 12: Simulation diagram of the wavelet encoder

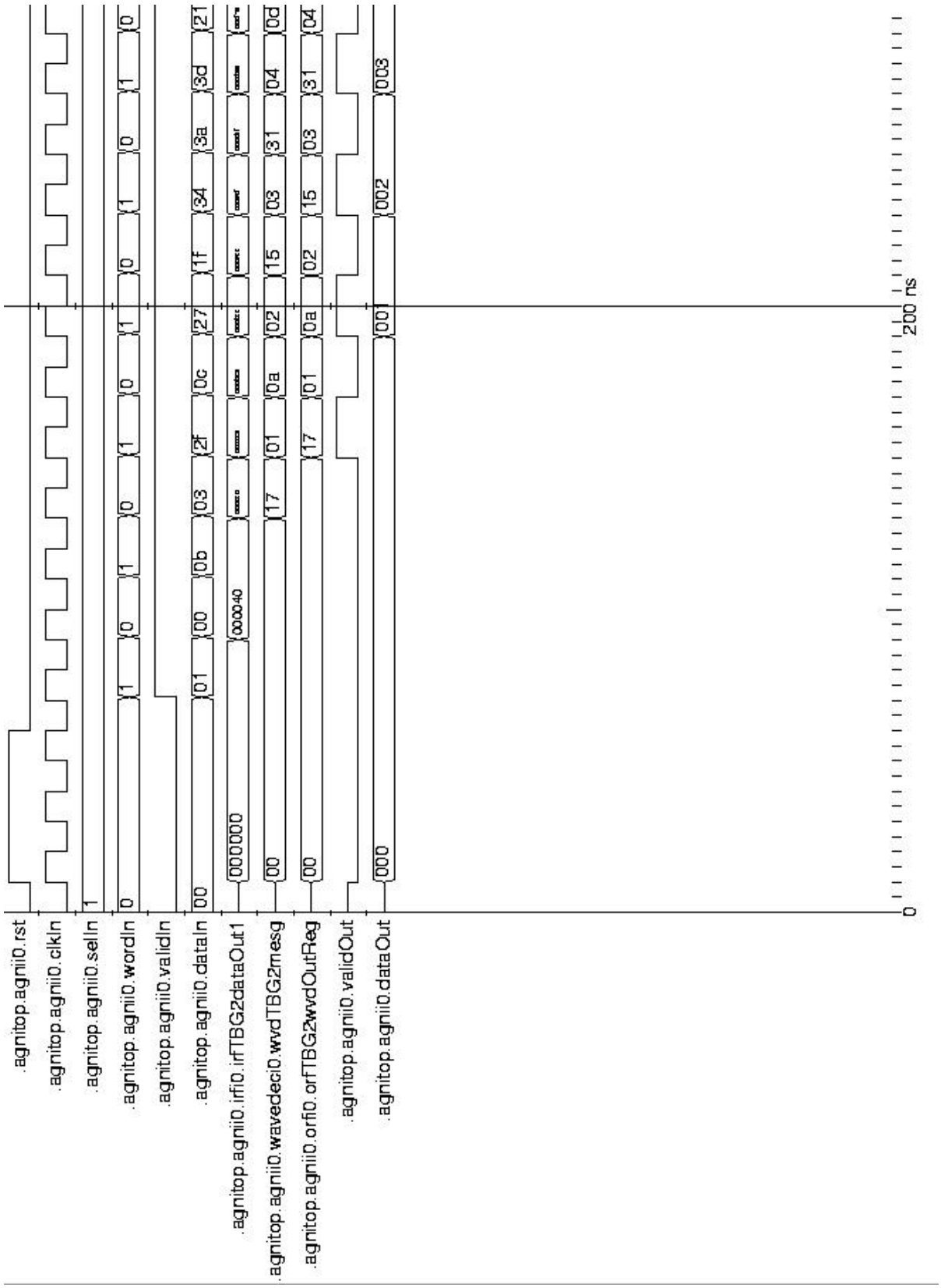

Figure 13: Simulation diagram of the wavelet decoder

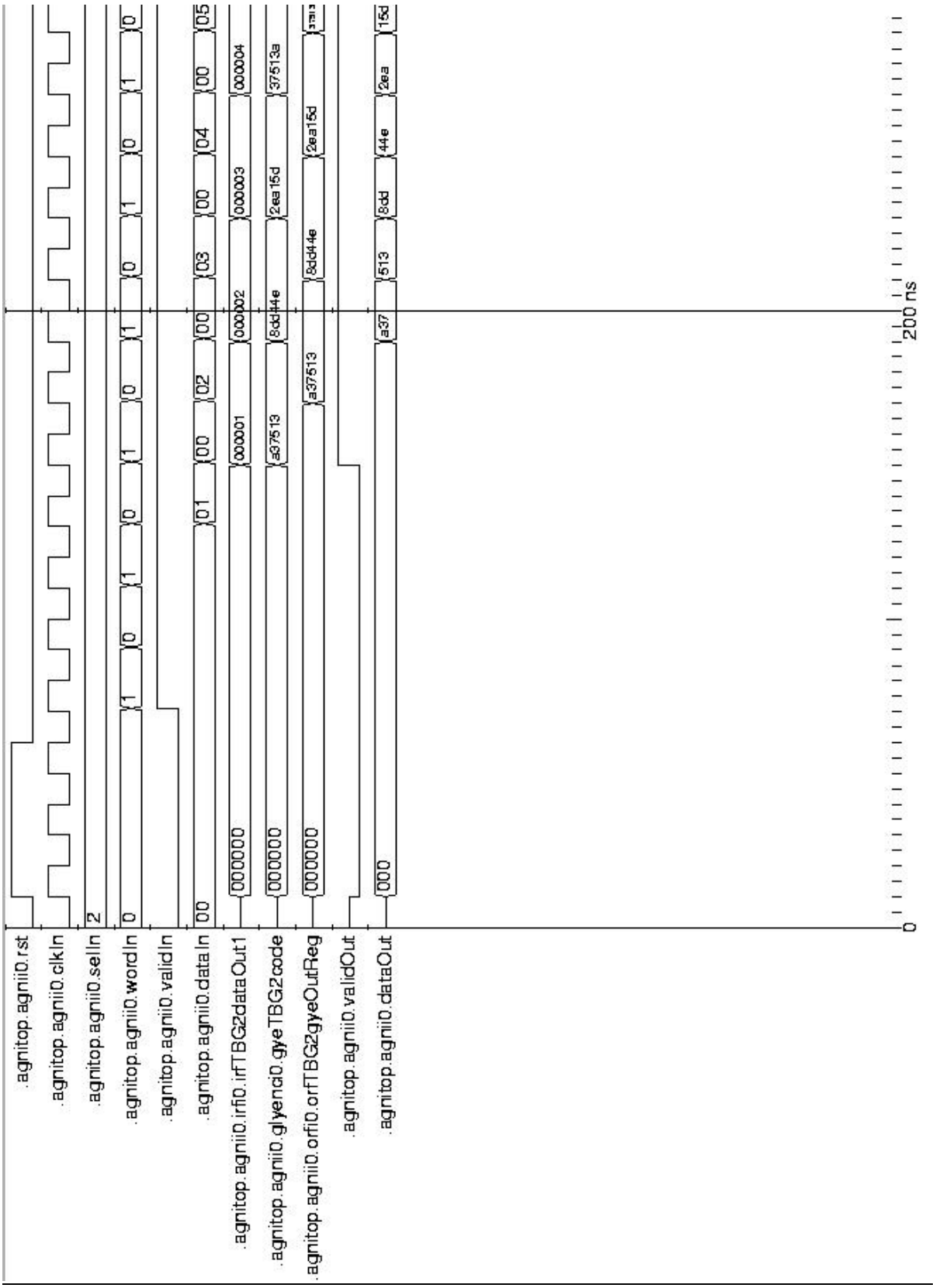

Figure 14: Simulation diagram of the golay encoder

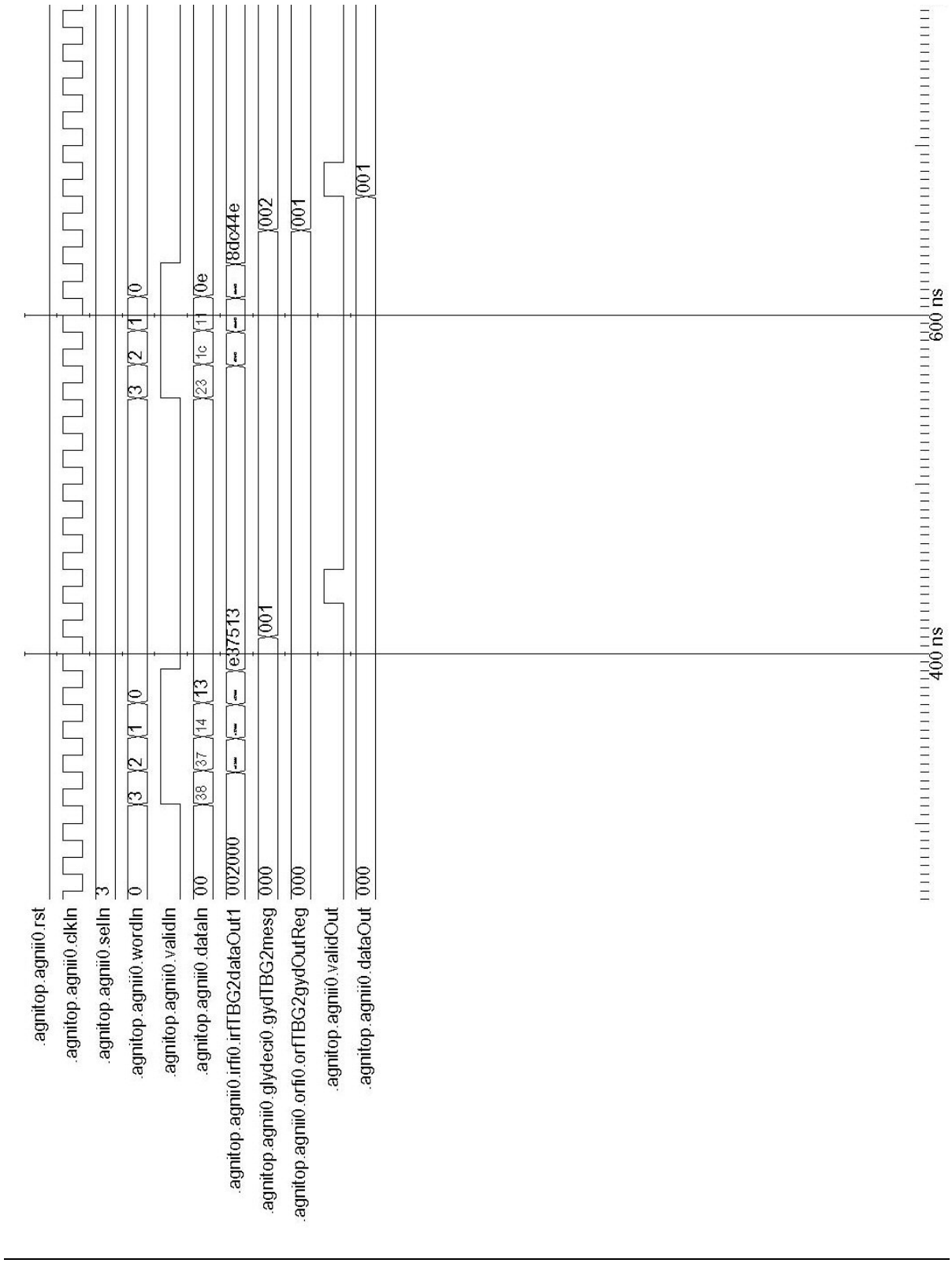

Figure 15: Simulation diagram of the golay decoder

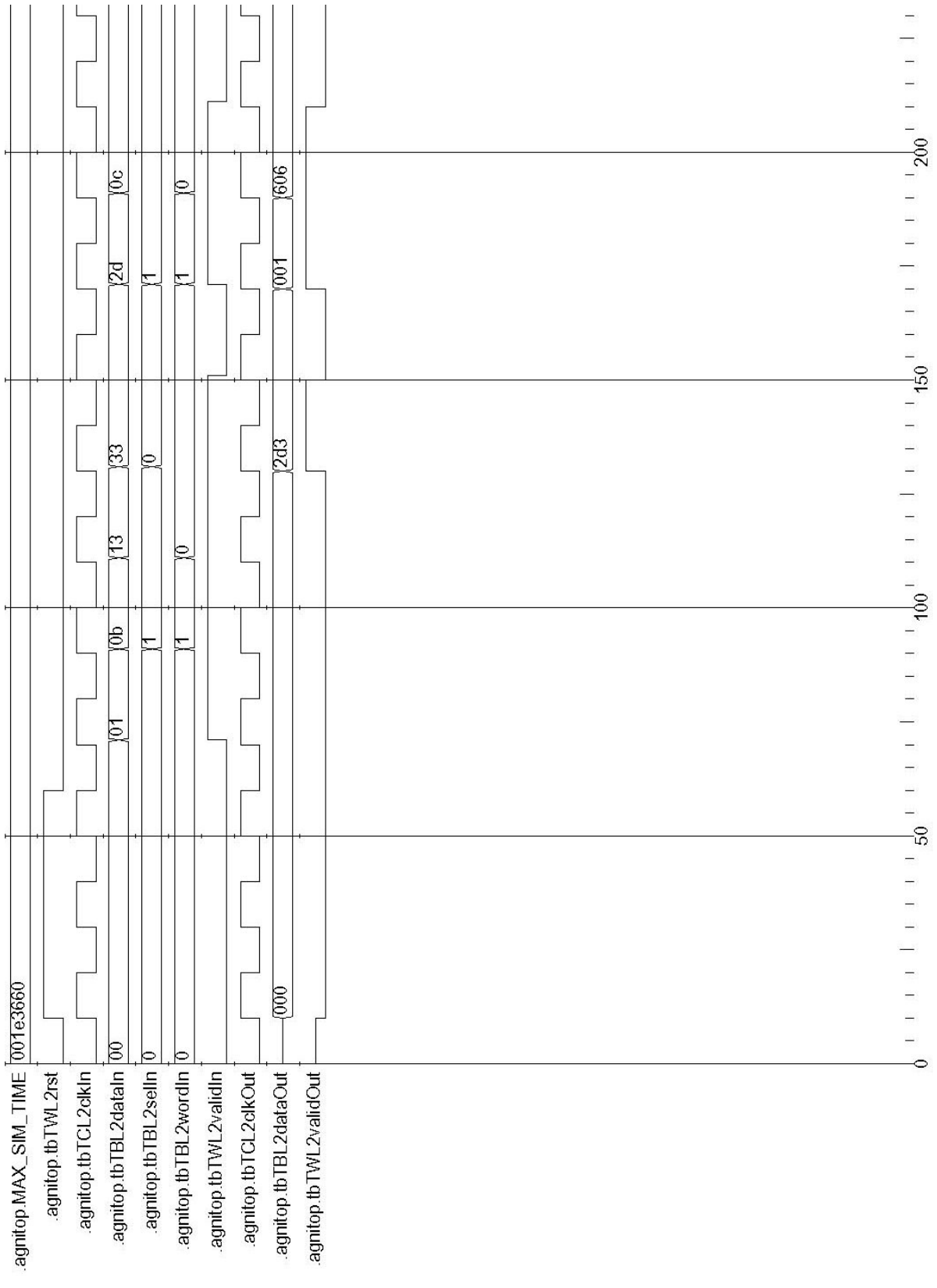

Figure 16: Simulation diagram for dynamic switching between wavelet encoder and wavelet decoder

3.3 Logic synthesis using Synopsys tools

The synthesis of the RTL design is done using the Synopsys Design Compiler [4]. The following steps are carried out in this process:

(a) A Synopsys setup file, .synopsys_dc.setup, is created.

(b) A synthesis script is created to read in the Verilog RTL code, to set operating conditions and optimization constraints, to insert I/O pads and to perform compilation.

(c) The synthesized netlist is ungrouped, i.e, all hierarchy is removed, and the netlist is written out along with preliminary timing information.

The target library for the synthesis process is Artisan's TSMC 0.25 μm CMOS library. This library has been obtained through joint work with Prof. John Uyemura. This library is characterized for slow, typical and fast parameters. The parameters corresponding to a fast process has been used in this design so as to obtain the best case performance estimates.

The synthesis results are obtained as Synopsys database (.db) format, Verilog netlist, delay Standard Delay Format (SDF) and constraint SDF. These delays, i.e, delay SDF and constraint SDF are based on a wire load model available through Artisan's library and represent an approximation to the final delays in the design. The Verilog netlist and delay SDF are used for timing simulations. The constraint SDF is used as input to the Place and Route tool.

3.4 Logical verification and timing simulations using MODELSIM

Validation and timing simulations of the synthesized netlist is done using Mentor Graphics' MODELSIM simulator. The following steps have been carried out in this process:

(a) The Verilog models for the gates, provided by Artisan, the synthesized netlist and standard SDF, obtained from the synthesis step, are set up for simulation.

(b) A testbench is set up and the design is compiled from the lowest module of the hierarchy to the top module.

3.5. Floor Planning, Automatic Place and Route using Cadence Silicon Ensemble

The Silicon Ensemble (SE) tool from Cadence [5] performs Place and Route. The Verilog netlist and the constraint SDF, obtained from the synthesis step, along with Verilog models for the standard cells, library and timing data provided by Artisan are given as input to SE. Library data is provided as a LEF (Library exchange format) file. Timing data is provided as TLF (Timing Library Format) files. The TLF files are not input directly to the tool; rather the file is specified in a GCF (General Constraints Format) file, along with operating conditions and miscellaneous information. The following steps are then carried out to obtain a physical layout of the design:

(a) Floorplanning**:** This step decides the approximate area of the chip, the amount of space for the core cells and the Input/Output (I/O) cells.

(b) I/O cell placement: I/O signal pads, VDD/VSS pads and corner pads, which are input to the tool as a Design Exchange format (DEF) file, are placed on the periphery of the die.

(c) Plan power: An outer VDD ring and an inner VSS ring are built around the core cell area and vertical metal stripes are evenly spaced across the core area. Choosing the right number of stripes to distribute power is critical to eliminating power routing problems within the core.

(d) Timing driven placement of core cells: Automatic placement of core cells is done using the SDF constraints for the paths in the design.

(e) Clock Tree Insertion: This step builds a tree of clock buffers so that all clocked elements (flip-flops) receive the clock at exactly the same time. This is critical to remove or lower clock skews in the design.

(f) Power routing: Power routing connects the VDD/VSS pins of cells to the VDD/VSS nets.

(g) Clock routing: This step is used to route clock to all cells in the design.

(h) Signal routing: Global and final routings are carried out simultaneously. Global routing assigns a list of routing regions (channels) for interconnects without specifying their actual geometric layout. Final routing specifies the geometric layout for the interconnects within their assigned regions.

The physical layout of the design is obtained at this stage as a graphic data system (GDS) file, which is clock routed, power routed and signal routed. This followed by verification of the obtained layout.

3.6. Layout verification

The physical layout is verified for timing and connectivity, geometry and antenna errors, as presented in the following subsections.

3.6.1. Post route timing analysis

Timing analysis can be carried out at several points during the place and route. Typically it is done after clock tree insertion, and also after the clock and signals have been routed. This serves as a check that the design still meets the timing goals. The design has been able to successfully meet a clock period of 5.2 ns.

3.6.2. Verification of connectivity errors

This is used to check the design for opens, antenna loops, partial routing and other connectivity problems among interconnects in the design. The design has been able to successfully pass this check.

3.6.3. Verification of geometry errors

This is used to check the design for width, spacing and internal geometry of cells and wiring between them. There were no geometry errors in the design.

3.6.4. Verification of antenna errors

This step reports pins in the design which have a violation due to the process antenna effect. The antenna effect is a side-effect of plasma-based etch processes, which are dominant because these processes can achieve very fine feature sizes. Plasma etchers or ion implanters can induce charge on a conductor, such as a metal layer, which is not connected to any diffusion region(s) but is only connected to gates. The conductor acts like an antenna collecting charges, and the accumulated charges may result in oxide breakdown of gates (pins) [6]. The check reported 18 antenna errors in the design.

The antenna errors in the design have been fixed using a combination of two techniques.

(a) One way is to break metal layers into small pieces so they can not build enough charge to destroy transistor input gate oxides. From a place and route CAD software point of view, this can be done by manually changing the routing of metal layers on those interconnects which have exceeded a certain critical length [7].

(b) Another way to remove the antenna effect is by inserting protection diodes to antenna violation metals (the metals are typically chip interconnect using Aluminum or, in some processes, Copper). Electrical charges on metals connecting to diodes will discharge through the diode diffusion layer and substrate. Diodes are normally placed as close to input pins as possible for the protection of input gate oxides. This has been done in Silicon Ensemble (SE) by setting the appropriate environment variables during the routing process [7].

At the end of the verification step, the design successfully reached a clock period of 5.2 ns and reported no connectivity, geometry or antenna errors. The area of the chip has been estimated to be 6.03 mm^2. The design was output as a GDS file and sent to MOSIS [8] for fabrication on November 1, 2002, for the 0.25μm TSMC run.

3.6. Implementation by MOSIS

The design successfully passed the verification steps done by MOSIS. The verification steps include *short*, *overlap*, *connectivity*, *geometry*, *antenna*, *electromigration* and *voltagedrop*. The physical layouts, prior to fabrication (left) and after fabrication and packaging (right), are shown in Figure 17. The design has been implemented with the following three packaging options: Dual in line package (DIP), Quad flat package and unpackaged die. The unpackaged die could be gold bumped for flip chip applications, as an extension of this work.

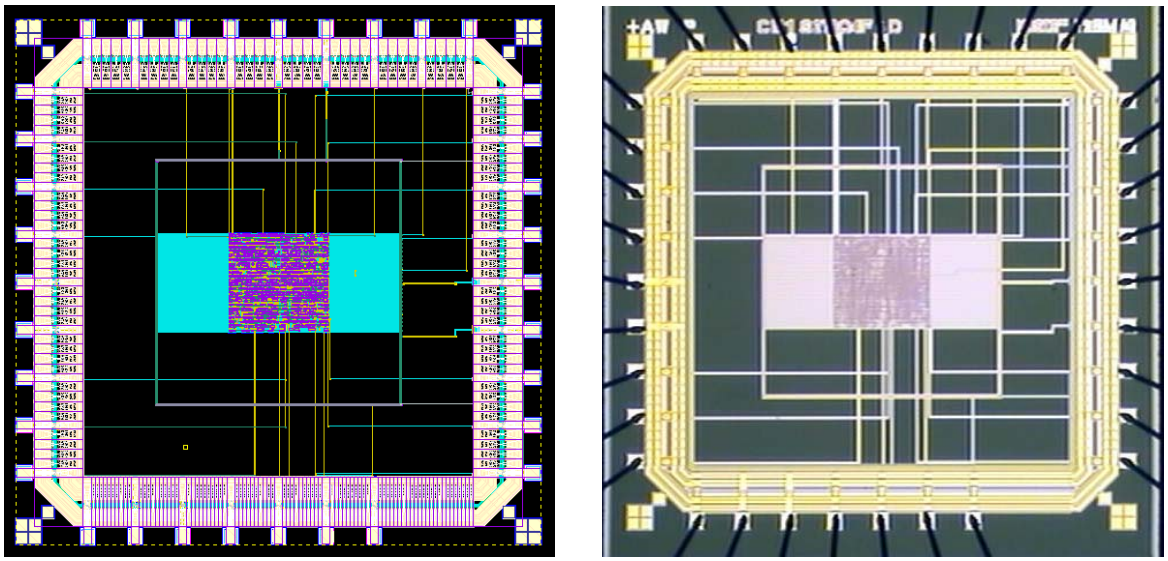

Figure 17: Physical layout prior to fabrication (left, given by MOSIS) and a chip layout with wire connections to the DIP after fabrication (right, taken at Georgia Tech)

4. TESTING THE INTEGRATED CIRCUIT (IC)

The fabricated chip has been tested using an HP 83000 Digital IC Test system (Courtesy of Dr. David Keezer, Professor, School of Electrical and Computer Engineering, Georgia Institute of Technology). A snapshot of the test system is shown in Figure 18. This test system provides flexibility for setting individual pin level and timing information as required by testing of complex, high speed IC's. An appropriate Printed Circuit Board (PCB) for the DIP was made as shown in Figure 19 by soldering a DIP test socket onto the PCB and then making the connections according to the DIP bonding diagram, provided by MOSIS. This PCB, referred to as the device under test, is connected to the test head, which connects I/O boards inside of the test system and the PCB with the DIP. The test head supports up to 512 pins. Each pin corresponds to a channel on an I/O board. The test system can support up to 32 I/O boards, each of which provides 16 bi-directional channels. A snapshot of the PCB connected to the test-head is shown in Figure 20. The test system provides data rate of up to 660 MHz. The test system has a mouse-controlled user interface that allows for testing and execution (Figure 21). The following sub-sections provide experimental details and result of the testing process.

Figure 18: Snapshot of the HP83000 Digital IC Test System

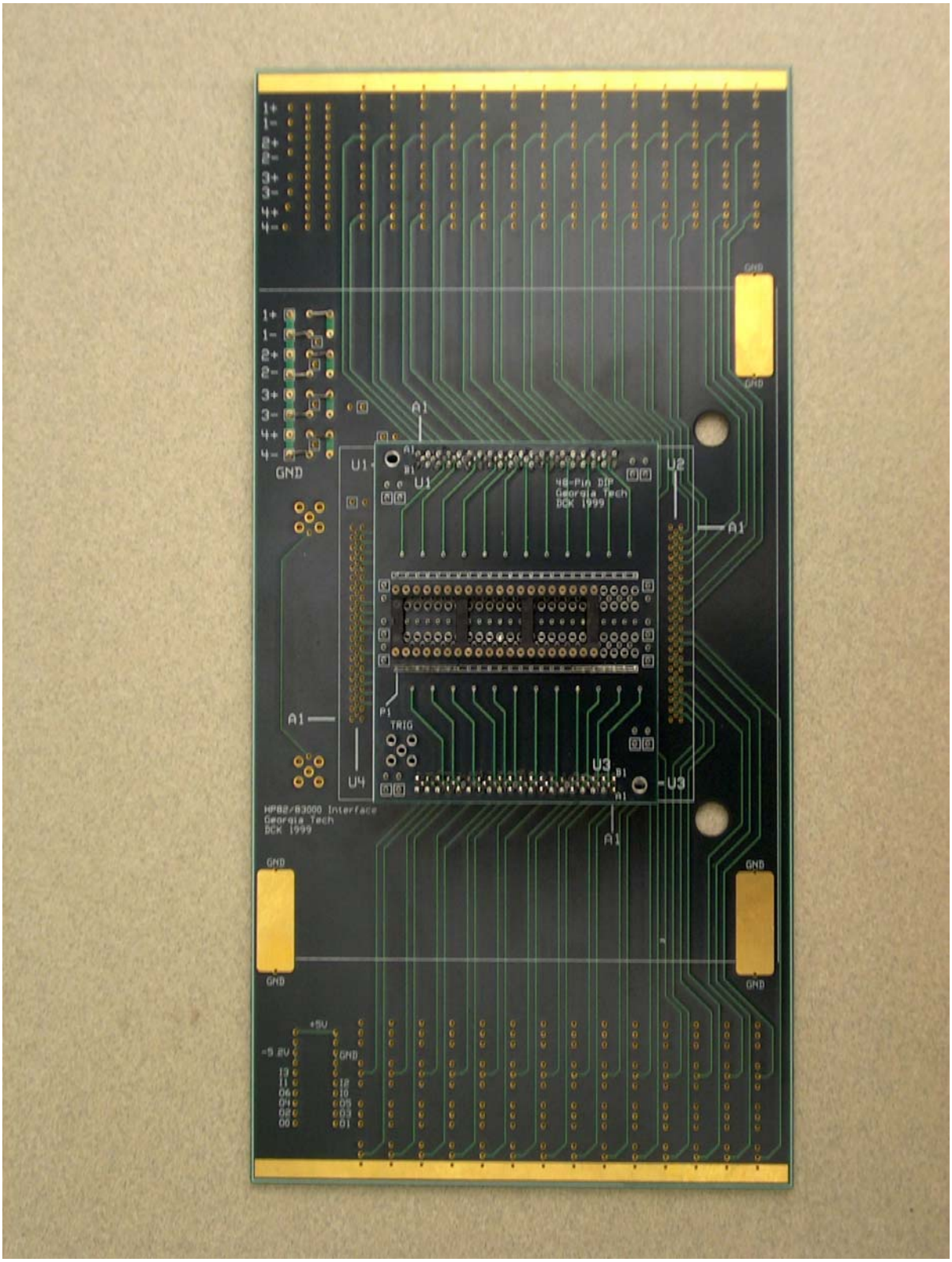

Figure 19: Snapshot of the printed circuit board

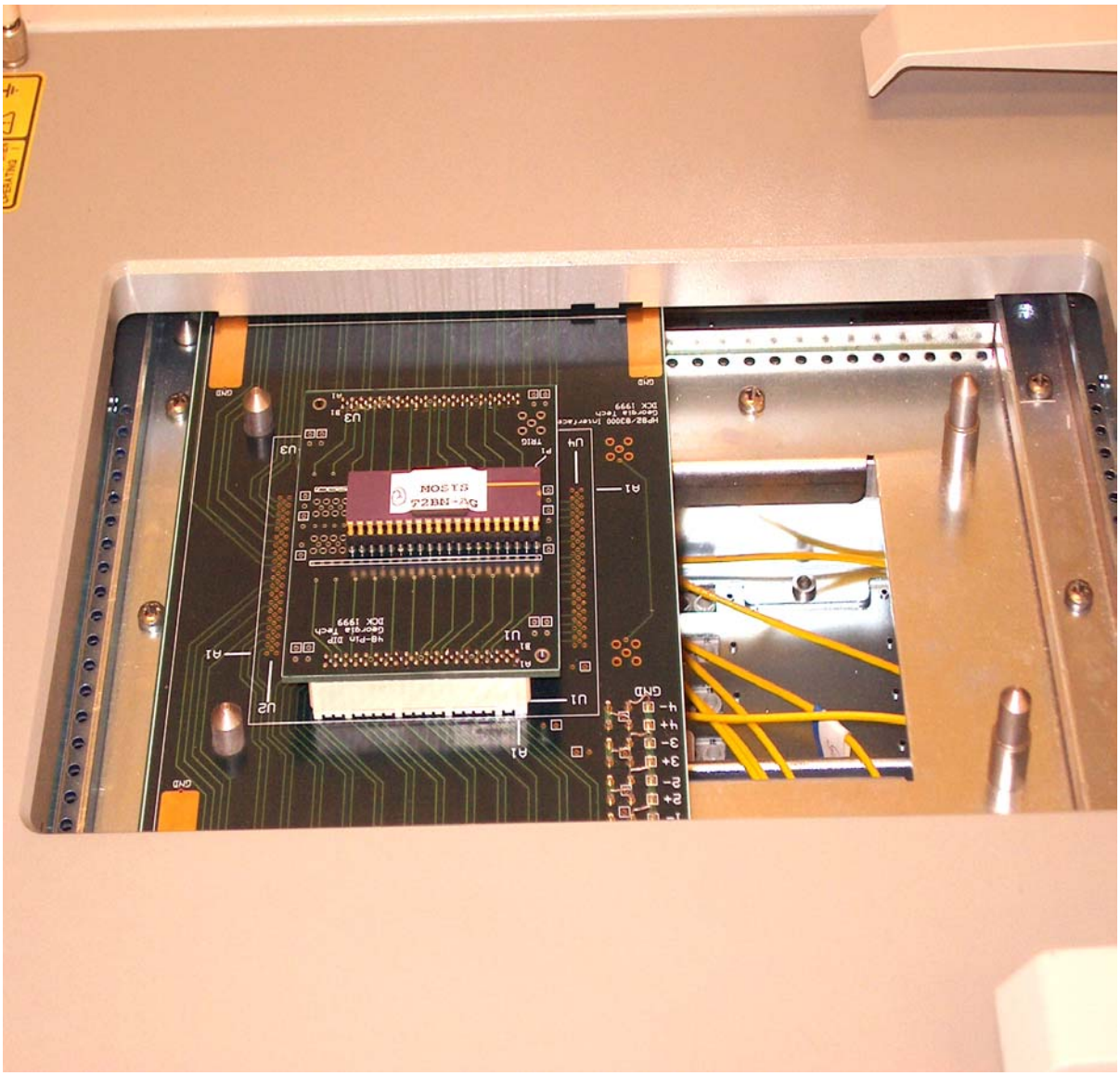

Figure 20: Snapshot of the PCB (holding the chip) mounted onto the test head

Figure 21: Computer interface of the HP 830000 Digital IC test system

4.1 Experimental details

A test program has been developed for checking the functionality of the device and measuring the total delay. The test program includes the following steps:

(a) Pin Configuration: The first step in developing a test program is pin configuration. This was used to match the input, output and device power pins to that of the tester channels as per the connections made. This is a crucial step as this specifies the tester resources to be used with the corresponding test pins.

(b) Level Setup: The next step is to fix the low and high voltage levels for each pin, which is configured. A low level voltage of 0V and a high level voltage of 3.3V have been used.

(c) Timing Setup: The timing setup is used to specify master clock period, waveforms and edge timing. A Return to Zero (RZ) waveform for clkin input pin and a Do not Return to Zero (DNRZ) format for the other input data pins (rst, selin, wordin, datain, validin) have been specified. An edge sampling has been specified for the outputs.

Test Vectors: The test vector is the input pattern that is supplied as input to the device under test. Subsets of test vectors have been randomly selected for the wavelet encoder/decoder and golay encoder: 5 for the wavelet encoder, 5 for the wavelet decoder, 5 for the golay encoder and 2 for the golay decoder. The output from the device is sampled for each test vector and checked for conformity with the desired result.

4.2 Results

The wavelet encoder, wavelet decoder, golay encoder and golay decoder have been successfully tested for their functionality at a clock period of 6.9 ns or a speed of 145 MHz. The setup time for the inputs is about 2 ns, and the delay on the outputs is approximately 5 ns. This measures closely to the speed achieved in simulation taking into account the delay introduced by the pins of the package. The maximum time of flight delay caused due to the parasitics of the leads of the DIP 40 package, as given by MOSIS, is about 0.21 ns. Figure 22 illustrates the parasitics arising due to the leads in the DIP 40 package. The maximum speed obtained in simulations, as reported earlier, is about 5.2 ns or 192 MHz. Taking into account the delay of 0.21 ns introduced by the package, the theoretical maximum speed achievable by the circuit is about 185 MHz, which is approximately 30% higher than the speed of 145 MHz that we have obtained in testing. One of the main reasons of the difference is a calibration error. We used manual calibration that can possibly have ±1 ns

error bound. Therefore, our simulation clock period can be adjusted up to 5.9 ns. The clock period can be increased but that is a very rare case in our testing system. Therefore, the upper bound of the clock period can reach 5.9 ns (170 MHz).

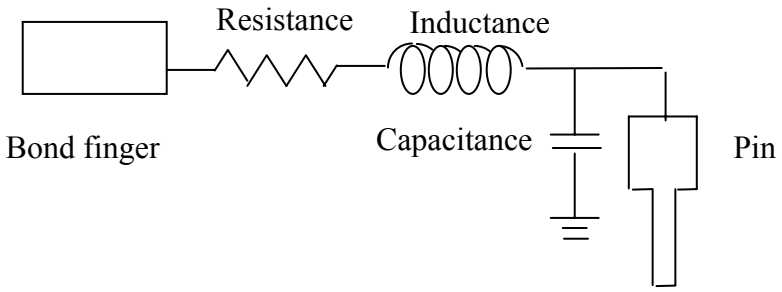

Figure 22: Parasitics arising due to the leads of DIP 40 package

5. COMPARISON OF SPEED OF THE CHIP IN TSMC 0.18 μm AND TSMC 0.25 μm TECHNOLOGIES

An estimate of an achievable chip speed in 0.18 μm technology is obtained by synthesizing a Fanout of 4 (FO4) inverter in TSMC 0.18 μm and TSMC 0.25 μm technologies in Synopsys Design Compiler and estimating the delays. The delay for the FO4 inverter in TSMC 0.18 μm technology is found to be 0.06 ns, while in the case of TSMC 0.25 μm technology, it is found to be 0.11 ns. As mentioned in Section 4.2, the speed of the chip, fabricated in TSMC 0.25 μm technology, is 145 MHZ as empirically measured in testing. Therefore, the theoretical speed of the chip in TSMC 0.18 μm technology can be estimated to be
(0.11 * 145) / 0.06 = 265.83 MHz.

6. Conclusion

We have presented an implementation of wavelet and golay error control codes, which has been fabricated by MOSIS in TSMC 0.25 μm technology. To the authors' best knowledge, this is the first silicon implementation of the wavelet encoder/decoder and the wavelet-based golay encoder/decoder. The fabricated chip has been successfully tested for its functionality at a high speed of approximately 145 MHZ. Since the input datain[5:0] has 6 bits and can be clocked in every clock cycle, this chip can achieve a data transfer rate of 870 Megabits/second.

7. REFERENCES

[1] F. Fekri, S. W. Mclaughlin, R. M. Mersereau and R. W. Schafer, "Decoding of half-rate wavelet codes; golay code and more," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP '01)*, Vol. 4, pp. 2609-2612, 2001.

[2] Artisan Inc., http://www.artisan.com

[3] Mentor Graphics Inc., http://www.mentor.com

[4] Synopsys Inc., http://www.synopsys.com

[5] Cadence Inc., http://www.cadence.com

[6] W. Maly, H.T. Heineken, J. Khare, P.K. Nag, P. Simon and C. Ouyang, "Design Manufacturing Interface: Part II - Applications," *Proceedings of Design for Manufacturability – Embedded Tutorial, (DATE 98)*, pp. 557-562, 1998.

[7] Source Link, Cadence, Inc., http://sourcelink.cadence.com

[8] MOSIS, http://www.mosis.org

[9] F. Fekri, S. W. Mclaughlin, R. M. Mersereau and R. W. Schafer, "Double circulant self-dual codes using finite field wavelet transforms," *Springer Verlag Lecture Notes in Computer Science (LNCS); Applied Algebra, Algebraic algorithms and Error-Correcting Codes*, pp355-364, 1999.

[10] F. Fekri, S. W. McLaughlin, R. M. Mersereau, and R. W. Schafer, "Decoding of half-rate wavelet codes; Golay code and more," *Proceedings of IEEE Int. Conf. Acoustics, Speech, and Signal Proc*., vol.4, pp. 2609 –2612, May 2001.