

VHDL modeling and synthesis of scramblers and CRC-processors for the B-ISDN

J.Carlos Calderón, José M. Tapia, Enric Corominas, Lluís París¹
Centre Nacional de Microelectrònica, CNM-CSIC.

¹Universitat Autònoma de Barcelona

Campus Universitat Autònoma de Barcelona 08193 Bellaterra, Barcelona, SPAIN

This paper describes the design a set of critical functions performed by a B-ISDN terminal equipment, with a SDH-based physical layer and an AAL of type 3/4. These functions are: synchronous and self-synchronizing scrambling / descrambling, CRC generation / check and error-correction based on Syndrome calculation. The high data-rate at the B-ISDN dissuades the use of simple LFSRs and a more complex parallel implementation must be used. The use of VHDL synthesis tools allows to design these functions in a simple and direct way.

I. Introduction

The future broadband ISDN (B-ISDN) will allow flexible and efficient integration of a wide variety of services such as video, data and voice [1]. The Asynchronous Transfer Mode (ATM) has been chosen to support them. It consists of fixed-size cells of 53 octets that are transmitted through the network with little delay and loss probability. At present, B-ISDN also relays on the Synchronous Digital Hierarchy (SDH) as a transmission system. The SDH hierarchy has a lowest rate of 155.52 Mbit/s which can be multiplexed in higher levels (622.08 Mbit/s, ...). The B-ISDN protocol reference model also provides a layer called ATM Adaptation Layer (AAL) which is intended to deal with different classes of services. Nowadays four AAL types are defined: AAL 1, AAL 2, AAL 3/4 and AAL 5.

The SDH frame for the 155.52 Mbit/s SDH-based interface is byte-structured and consists of 9 rows and 270 columns (see Fig. 1). The frame repetition frequency is 8 KHz: $9 \times 270 \times 8 \text{ KHz} = 155.52 \text{ Mbit/s}$. The frame is broken down into two basic parts: the frame overhead (the first 10 columns) and the frame payload (the remaining 260 columns). The overhead bytes provides functions such as framing, operation, and maintenance. The payload bytes carry user information in ATM cells: the payload capacity is entirely used for ATM-cell mapping. The ATM cell, consisting of a 5 octet header, for routing and transmission purposes, and a 48 octet information field, allow efficient multiplexing of continuous and bursty traffic. AAL is an additional layer above ATM which adapts the characteristics of the ATM network to those required by the user of the network. The type 3/4 is used to provide connectionless and connection oriented data transfer services.

The purpose of this paper is to show how to design a set of functions performed by a B-ISDN terminal equipment by using VHDL-based synthesis tools. These functions are: scrambling, descrambling, CRC generation, CRC checking and error-correction based on syndrome calculation. These functions can be easily implemented using linear feedback shift registers (LFSR). However, for the high data-rates of the B-ISDN the traditional serial-defined methods are too slow, and a faster method must be used. A parallel calculation approach meets the

performance requirements, and also simplifies the overall system design. This work belongs to a project called UNICORN where a high speed internetworking unit is being developed. It will interconnect low and medium speed networks using the B-ISDN services. An ATM Network Adapter Board has been developed, implementing the three lower layers of the B-ISDN reference protocol model as specified by ITU, i.e. the SDH based physical layer, the ATM layer and the AAL 3/4.

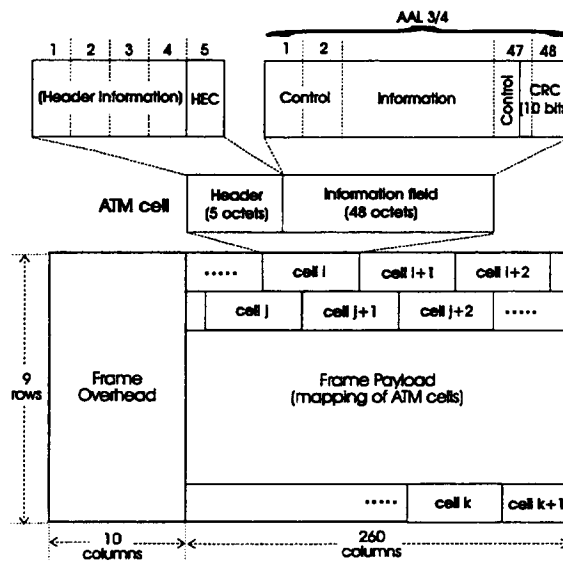


Fig 1. SDH frame and ATM cell formats.

The remainder of this paper is organized as follows: a serial approach for scrambling and CRC-processing performed by a ATM NAB is given in section II, basics of parallel approach are described in section III, synthesizable VHDL models for scrambling and CRC-processing are show in section IV, and finally, the error-correction process is detailed in section V.

II. Scrambling and CRC functions in the B-ISDN. Serial approach

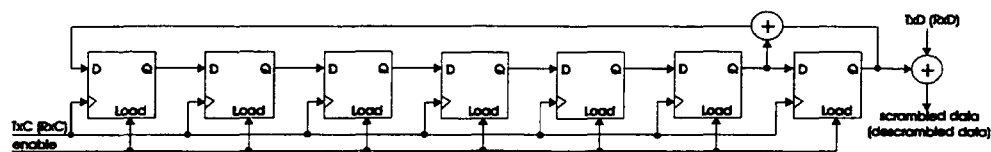
In order to avoid long strings of ones and zeros and allow clock recovery at receivers the SDH frame is scrambled in the transmitter side (before the electrical-to-optical conversion), and descrambled at the receiver side (after the optical-to-electrical conversion). The only bytes left unscrambled are the first 9 octets of the first row. The reference model standardizes a synchronous scrambler [2] with polynomial x^7+x^6+1 . The serial synchronous scrambler (see Fig. 2.(a)) is implemented by exclusive-ORing a pseudorandom bit pattern with the data. The same circuit is used in both the transmitter and the receiver sides.

The fifth octet of the ATM cell header is the header error control (HEC) sequence (see Fig. 1), which is processed by the physical layer [3]. The HEC value has the following well-known CRC-definition. Every ATM cell transmitter calculates the HEC value across the first four octets of the cell header and inserts the result in the fifth octet (HEC field). The transmitter generates the HEC value as the remainder of the division (module 2) by the generator polynomial x^8+x^2+x+1 [3] of the product x^8 multiplied by the content of the header excluding the HEC field. The receiver obtains the syndrome, defined as the remainder of the division (module 2) by the same generator polynomial, of the entire 5-octet header.

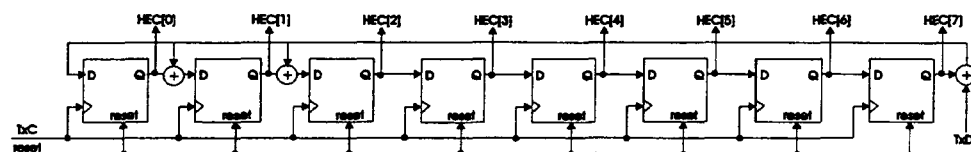
This HEC code is capable of correcting single-bit errors and detecting certain multiple bit-errors in the ATM cell header (both capabilities will be used by the equipment receiving ATM cells). To do this, the receiver usually memorize the N syndrome values ($N = 40$ bits) that corresponds to single-bit errors and follows a simplified correction algorithm: (a) if the

calculated syndrome is null, then the incoming header has no errors, (b) if the calculated syndrome is equal to the memorized syndrome-*i*, then the incoming header has a correctable error in position-*i*, (c) otherwise, the header has non-correctable multiple-bit errors [4, 5].

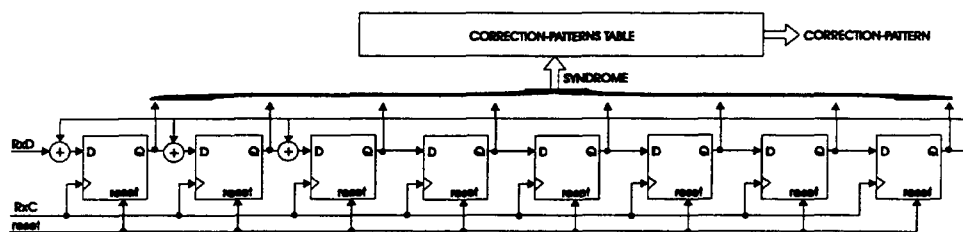
In the serial approach, the transmitter implements the HEC generation using the encoder circuit shown in Fig 2.(b), and the receiver calculates the syndrome and corrects the header as shown in Fig 2.(c).



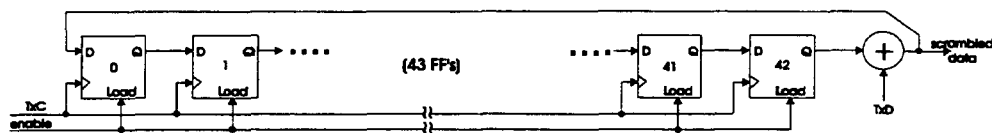
(a). Transmitter/receiver synchronous frame scrambler



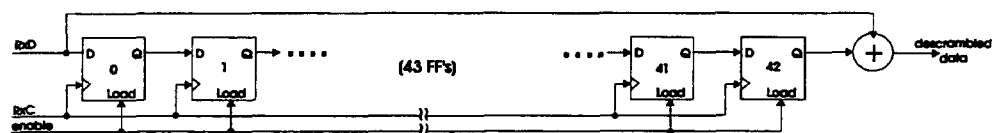
(b). HEC generator (encoder circuit)



(c). Syndrome calculator (decoder circuit) & pattern-correction table



(d). Self-synchronizing scrambler



(e). Self-synchronizing descrambler

Fig.2. Scrambling and CRC-processing serial implementations

The HEC checking is also used to accomplish the cell delineation: cell delineation is the process which allows identification of the cell boundaries. The method recommended by ITU [3] is based in the correlation between the header bits to be protected (the first 4 octets of the cell header) and the HEC field. This delineation method could fail if the header HEC correlation

were imitated in the information field of ATM cells. This might be effected by a malicious user or inadvertently by an application that accidentally uses the same generator polynomial. To overcome such difficulties the information field contents will be scrambled using a self-synchronizing scrambler with the polynomial $x^{43}+1$. The self-synchronizing scrambling has the following important feature: after a synchronism-loss (cell delineation loss, or bit-errors in the information field), the receiver circuitry will recover automatically the synchronism with the transmitter circuitry. Fig. 2.(d) and Fig. 2.(e) show respectively the serial self-synchronizing scrambler and descrambler circuits.

AAL 3/4 also uses a CRC (see. Fig. 1). This CRC is a ten-bit sequence, protecting the information field, and used for error-detection (no correction). The CRC calculation is very similar to the HEC calculation (described above), using the generator polynomial $x^{10}+x^9+x^5+x^4+x+1$ [6].

III. The parallel-calculation approach. VHDL suitability

The serial implementation has two important problems: (a) The serial circuitry works at the line rate. For high-speed communications networks, as B-ISDN (155.52 Mbit/s, 622.08 Mbit/s, ..), the serial approach is too slow, forcing to use an expensive high-speed technology. (b) In B-ISDN circuits, data is processed in byte-format: the parallel approach also simplifies the overall system design. The serial implementation can be broken in two parts: a flip-flops block (the sequential circuitry) storing the present state (PS) and working with the serial clock, and a XOR-PLANE block (the combinational circuitry) calculating the next state (NS) and optionally the output (O), as a function of the present state and the present input (I). The flip-flops block starts at state S_0 , and the XOR-PLANE calculates the sequence $S_1, S_2, S_3, S_4, \dots$.

$$\begin{aligned}
 S_1 &= NS (S_0, I_0) \\
 S_2 &= NS (S_1, I_1) = NS (NS (S_0, I_0), I_1) = NS^{(2)} (S_0, I_0, I_1) \\
 S_3 &= NS (S_2, I_2) = NS (NS (NS (S_0, I_0), I_1), I_2) = \\
 &= NS^{(3)} (S_0, I_0, I_1, I_2) \\
 (\dots\dots) \\
 S_N &= NS (S_{N-1}, I_{N-1}) = \dots = NS^{(N)} (S_0, I_0, I_1, I_2, I_3, \dots, I_{N-1})
 \end{aligned}$$

Fig 3. Next state computation

The parallel approach lies in a modified XOR-PLANE block calculating the $NS^{(N)}$ function [7], and optionally the $O^{(N)}$ function. Note that in a CRC calculation, $O^{(N)}$ is equivalent to $S_N = NS^{(N)}(S_0, I_0, I_1, \dots, I_{N-1})$. In the parallel implementation the flip-flops block is not modified, but it works with the parallel clock: "parallel clock frequency" = "serial clock frequency" / N. Fig. 4 shows a parallel circuitry scheme.

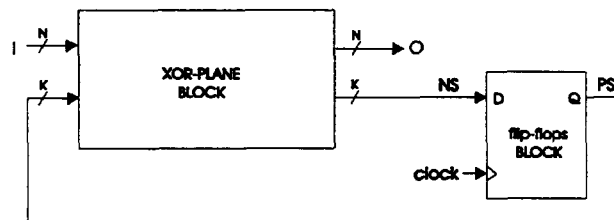


Fig. 4. Parallel circuitry scheme

A low-level solution is to calculate, manually or with a program (which has to be written first), the parallel XOR-PLANE equations, to simplify these, and then to introduce this result in the EDA-tool. Transforming these equations to their gate level equivalent is a tedious and error prone task. By using VHDL, a sparse description generate the same result in a direct way [7]. The basic idea is to elaborate the $NS^{(N)}$ and $O^{(N)}$ functions (the XOR-PLANE) in a recurrent way (as they are defined) using a for-loop structure (the loop-variable acts as the serial clock). Fig. 5 shows a VHDL description modeling a generic XOR-PLANE; constant K is the generator polynomial order, and constant N is the parallelism degree.

```

signal I, O : Std_Logic_Vector(N-1 downto 0);
signal PS, NS : Std_Logic_Vector(K-1 downto 0);
FLIPFLOPS: process(clear, ck)
begin
  if clear = '1' then PS <= "000...0";
  elsif ck'event and ck = '1' then PS <= NS;
  end if;
end process;
XORPLANE: process(I, PS)
variable PS_aux, NS_aux : Std_Logic_Vector(K-1 downto 0);
variable t : integer range 0 to N-1;
begin
  PS_aux := PS;
  U1: for t in 0 to N-1 loop
    NS_aux := f ( PS_aux, I(t) ); O(t) <= g ( PS_aux, I(t) );
    PS_aux := NS_aux;
  end loop;
  NS <= NS_aux;
end process;

```

Fig. 5. Parallel circuitry VHDL generic description

IV. VHDL modeling for synthesis of scrambling and CRC-processing Functions

This section shows the synthesizable VHDL models that parallelize the serial-defined functions explained in section 2. All these functions match the generic VHDL model presented in section 3.

Since, in our application, data is processed in byte format, the synchronous and self-synchronizing scrambling / descrambling functions have been created as 8-bit parallel machines.

The HEC generator also works in byte format. In exchange, the HEC checker (syndrome generator) has been created as a full-parallel machine (40-bit parallel machine), because the HEC checking is also used for the cell-delineation process [3]. In the full parallel machine the flip-flops block disappears, since the full parallel machine does not need to memorize intermediate states.

The AAL 3/4 CRC generator circuit has been created as a mixed 8-bit and 6-bit parallel machine, because the CRC field (10 bits) protects a non 8-bit multiple sequence (48 bytes - 10 bits = 46 bytes + 6 bits). Conceptually, such XOR-PLANE is achieved with a non-constant loop range, for loop_variable in 0 to N-1, where N can take two different values (8 or 6). This is not possible, since a range must be constant. The solution is to write (see below) two different for-loops (two different XOR-PLANES). In this way the synthesis-tool mixes and optimizes both XOR-PLANES obtaining an optimum result, because these XOR-PLANES share a lot of common

equations (remember his recurrent definition at section 3). The AAL 3/4 checker is a simple 8-bit parallel machine, very similar to the HEC checker (syndrome generator). Figures 7, 8, 9, 10 and 11 shows the synthesizable VHDL models for these blocks.

```

if lenght_eq_8 then
  loop8 : for t in 0 to 7 loop ..... end loop;
else
  loop6 : for t in 0 to 5 loop ..... end loop;
end if;

```

Fig. 6. AAL 3/4 CRC generation

```

process(I, PS, reset, enable)
variable t : integer range 0 to 7;
variable PS_aux, NS_aux : Std_Logic_Vector(6 downto 0);
begin
  PS_aux := PS;
  for t in 0 to 7 loop
    if (reset = '0' and enable = '1') then
      O(7-t) <= I(7-t) xor PS_aux(6);
    else O(7-t) <= I(7-t);
    end if;
    NS_aux(0) := PS_aux(5) xor PS_aux(6);
    NS_aux(6 downto 1) := PS_aux(5 downto 0);
  end loop;
  NS <= NS_aux;
end process;

```

Fig. 7. STM synchronous scrambler-descrambler

```

process(I, PS, enable)
variable t : integer range 0 to 7;
variable feedback : Std_Logic;
variable NS_aux : Std_Logic_Vector(42 downto 0);
begin
  NS_aux := PS;
  for t in 0 to 7 loop
    feedback := NS_aux(42) xor I(7-t);
    if enable = '0' then O(7-t) <= data(7-t);
    else O(7-t) <= feedback;
    end if;
    NS_aux(42 downto 1) := NS_aux(41 downto 0);
    NS_aux(0) := feedback;
  end loop;
  NS <= NS_aux;
end process;

```

Fig. 8. ATM cell payload self-synchronizing scrambler

```

process(I, PS, enable)
variable t : integer range 0 to 7;
variable NS_aux : Std_Logic_Vector(42 downto 0);
begin
  NS_aux := PS;
  for t in 0 to 7 loop
    if (enable = '1') then
      O(7-t) <= NS_aux(42) xor I(7-t);
    else O(7-t) <= I(7-t);
    end if;
    NS_aux(42 downto 1) := NS_aux(41 downto 0);
    NS_aux(0) := I(7-t);
  end loop;
  NS <= NS_aux;
end process;

```

Fig. 9. ATM cell payload self-synchronizing descrambler

```

process(I, PS)
variable t : integer range 0 to 7;
variable NS_aux, PS_aux : Std_Logic_Vector(7 downto 0);
begin
  PS_aux := PS;
  for t in 0 to 7 loop
    feedback := I(7-t) xor PS_aux(7);
    NS_aux(1) := PS_aux(0) xor I(7-t) xor PS_aux(7);
    NS_aux(2) := PS_aux(1) xor I(7-t) xor PS_aux(7);
    NS_aux(7 downto 3) := PS_aux(6 downto 2);
    PS_aux := NS_aux;
  end loop;
  NS <= NS_aux;
end process;

```

Fig. 10. ATM cell HEC generator

```

process(I)
variable t : integer range 0 to 39;
variable PS_aux, NS_aux : Std_Logic_Vector(7 downto 0);
begin
  PS_aux := "00000000";
  for t in 0 to 39 loop
    NS_aux(0) := I(39-t) xor PS_aux(7);
    NS(1) := PS(0) xor PS(7);
    NS(2) := PS(1) xor PS(7);
    NS(7 downto 3) := PS(6 downto 2);
    PS_aux := NS_aux;
  end loop;
  SYNDRO <= NS_aux;
end process;

```

Fig. 11. ATM cell HEC checker (syndrome generator)

V. VHDL modeling for synthesis of the correction-patterns table

In this section we describe the correction-patterns table model, because it is the only function that not match (in a direct way) the generic VHDL model presented in section 3. Fig. 12 shows how the correction-patterns table is used on the receiver circuitry. The *syndrome generator* (see above section) is a wholly paralleled implementation of the syndrome calculation algorithm; this combinational block process an entire cell header (REG1 to REG5 outputs) and generates the eight-bit syndrome. The receiver follows a simplified correction algorithm: (a) if the calculated syndrome is null, then the incoming header has no errors, (b) if the calculated syndrome is equal to the memorized syndrome-*i*, then the incoming header has a correctable error in position-*i*, (c) otherwise, the header has non-correctable multiple-bit errors. The *correction patterns table* memorize the N syndrome values that correspond to the N single-bit errors (N = 40 header bits) and generates the correction pattern to be applied on the first four header bytes. Fig. 13 shows the synthesizable VHDL model of the correction-patterns table.

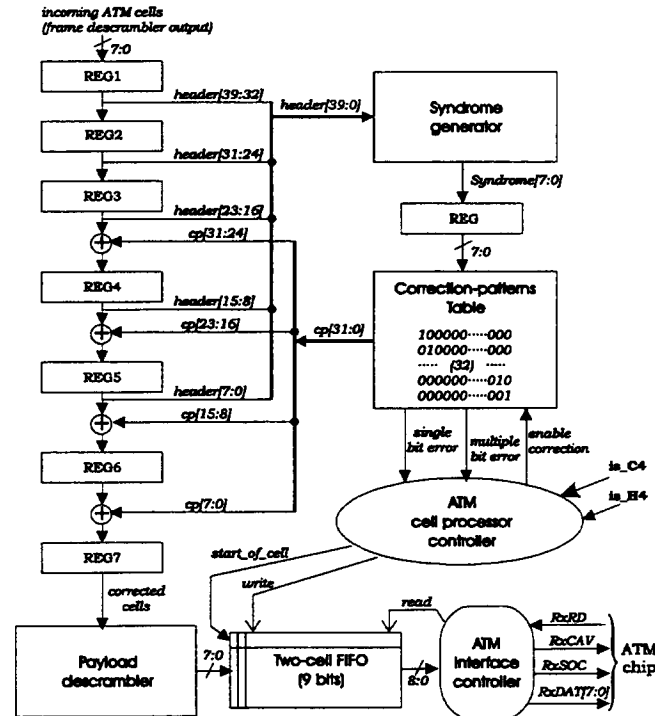


Fig. 12. Parallel header cell checking & correction

In order to understand this VHDL model it is necessary to take into account two basic properties:

- (a) an all-zeros sequence (i.e. the first four header bytes) generates a null CRC (i.e. the HEC),
- (b) any message (i.e. the entire header) with the same error(s) will generate the same syndrome.

Property (a) says that an all-zeros header (40 bits) is a correct message. Property (b) implies that any header with a single-bit error in position-*i* will generate the same syndrome that a header with only the bit-*i* equal to 1: 000000....0100...00000.

The designed VHDL model works as follows: generates the 40 possible syndromes, and compares the incoming syndrome with each of them. Two nested for-loops are used: the first

loop generates the 40 single-bit errors, and the second loop calculate the 40 corresponding syndromes.

The scrambling and CRC-processing functions were made in a Sun-SPARC-2 computer with 32 MB of RAM, and the synthesis-tool only took a few minutes to synthesize each circuit. It was not possible to synthesize the correction-patterns table in the same machine: there was no sign of it coming to an end. In exchange, using a Sun-SPARC-2 with 64 MB of RAM the correction-patterns table was synthesized taking a long time (approximately 1 hour). Fig 14 shows the generated schematic for the correction-patterns table.

```

CORRECTION_PATTERN_TABLE:
process(SYNDRO)
variable bit_error, t : integer range 0 to 39;
variable mult_error_aux : Std_Logic;
variable PS_aux, NS_aux : Std_Logic_Vector(7 downto 0);
begin
  mult_error_aux := '1';
  for bit_error in 0 to 39 loop
    PS_aux := "00000000";
    for t in 0 to 39 loop
      NS_aux(0) := Boolean_to_Std_Logic(t = bit_error) xor
        PS_aux(7);
      NS_aux(1) := PS_aux(0) xor PS_aux(7);
      NS_aux(2) := PS_aux(1) xor PS_aux(7);
      NS_aux(7 downto 3) := PS_aux(6 downto 2);
      PS_aux := NS_aux;
    end loop;
    if (bit_error <= 31) then
      PATTERN(31-bit_error) <=
        Boolean_to_Std_Logic(NS_aux = SYNDRO);
    end if;
    if (NS_aux = SYNDROME) then mult_error_aux := '0';
    end if;
  end loop;
  HEC_OK <= Boolean_to_Std_Logic(SYNDROME = "00000000");
  ERR_MULT <= mult_error_aux and
    Boolean_to_Std_Logic(SYNDROME /= "00000000");
end process;

```

Fig. 13. Correction-patterns table

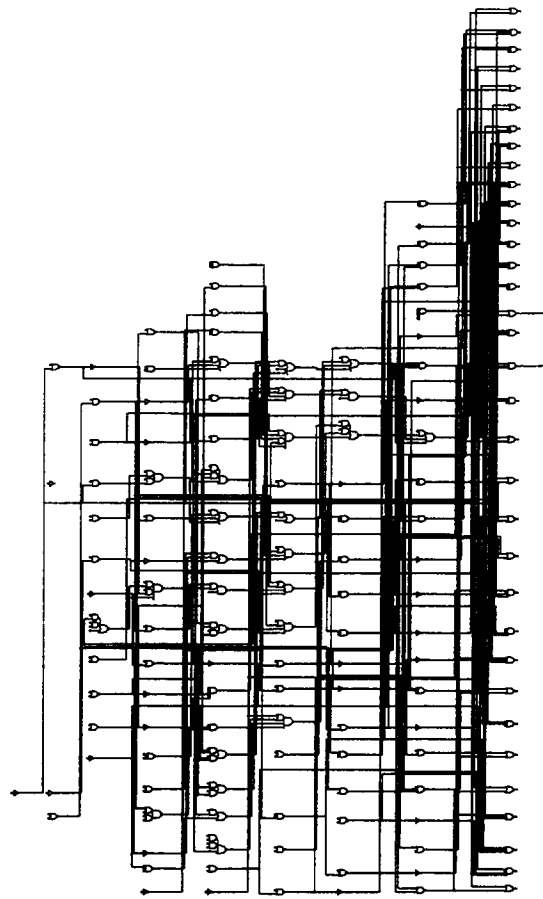


Fig. 14. Correction-patterns table schematic

References

- [1] R. Spears, "Broadband ISDN switching capabilities from a services perspective", *IEEE J. Select. Areas Commun.*, vol. 5, no. 8, pp. 1222-1230, Oct. 1987.
- [2] ITU-T Recommendation, G.709; Helsinki, 1993
- [3] ITU-T Recommendation, I.432; Helsinki, 1993
- [4] Alain Glavieux, "Theorie de l'information et du codage", Ecole Nationale Supérieure Des Telecommunications de Bretagne.
- [5] Lin and D. J. Costello, "Error Control Coding: Fundamentals and Applications". New York: Prentice-Hall, 1983.
- [6] ITU-T Recommendation, I.363; Helsinki, 1993
- [7] Olaf Geisler, Test and Measurement Systems, AUT E3 TA1A, Siemens AG, Berlin, Germany. "VHDL based synthesis of a high speed full parallel 32 bit CRC generator/checker". VHDL-Forum for CAD in Europe, Spring'93 Meeting.