SOOMRO AA, SADIQ M. SAIT, RAHMAN M•BIT-SLICE MICROPROCESSOR-BASED COMMUNICATIONS DECODER •MICROPROCESSORS AND MICROSYSTEMS 11 (10): 527-533 DEC 1987

Bit-slice microprocessor-based communications decoder

For communications decoding, bit-slice microprocessors can be cascaded to handle variable code sizes with fast throughput rates. **Amjad A Soomro, Sadiq M Sait* and Mushfiqur Rahman†** explain the operation of an Am2910-based system designed for this purpose

The hardware design of a bit-slice microprocessor-based realtime cyclic error-correcting communications decoder is presented. A microprocessor-based architecture is preferred because of its programmability, low cost and simplicity of design. To augment the throughput of the decoder for realtime decoding, the ALU word length is chosen to be equal to that of a code word and the decoding operation is accomplished in two steps, i.e. error detection and error correction. A buffer memory stores incoming blocks as more than one block may be received during a decoding cycle. The design is versatile: different decoding algorithms can be executed by changing the microprogram. Only simple changes in the design are necessary to decode words of longer block length.

microprocessors digital communications decoding

Cyclic error correcting codes are classes of codes in which every cyclic shifted version of a code word is also a valid code word. These codes can be represented by a generator polynomial. The cyclic property of these types of codes has made possible the design of a simple decoder structure using feedback shift registers.

Other classes of codes exist which are not cyclic but which are considered to be 'good' by virtue of their properties, which are code efficiency and error correcting capability, for example. Efficient decoding algorithms do exist for these codes but have remained in relative obscurity because their decoders require elaborate circuitry, offsetting the gains offered by the codes. Thus, cyclic codes have remained favoured over them.

In this paper a general microprocessor-based decoder is proposed. The reasons for using microprocessors in the design of decoders include programmability, simplicity of design, easy availability of microprocessors, and low cost of the devices.

Dammam Computer Center, Riyadh Bank, PO Box 2965, Dammam-31461, Saudi Arabia
*Department of Computer Engineering, King Fahd University of Petroleum and Minerals, KFUPM #673, Dhahran-31261, Saudi Arabia
†Department of Electrical Engineering, University of Petroleum and Minerals, KFUPM #377, Dhahran-31261, Saudi Arabia

In digital communications, the choice of a suitable code depends on various system constraints: channel bandwidth limitations, channel noise characteristics, data transmission rate, receiver complexity, cost effectiveness, etc. A bit-slice microprocessor-based architecture makes it possible to adapt the system with minimal changes to suit the requirements of a new code, or a different decoding algorithm, or both. Only if the new code word length is greater than the ALU size will the design require one or more ALU slices to be added along with associated changes in bus width, memory word size, etc. This is in contrast to hardware-oriented design which requires complete redesign of the system. If the decoding algorithm remains the same for the new code, the microprogram essentially remains unchanged except for minor changes in parameter values and input conditions. A new decoding algorithm will of course require a new microprogram.

Previous attempts at using microprocessors in realtime decoding were confronted with two major limitations. First was the inherent limitation of the processors then available to handle only words of fixed size. Codes with larger block lengths had to be partitioned for decoding purposes on these microprocessors. ALU operation could be performed on each partitioned block taking into account the effect of carry-over, etc., from the previous result. The number of transfers between memory and ALU in this procedure for very large word lengths severely limited the performance of such an approach. If the ALU word length is made equal to that of the code word, then a single operation can be performed on a code word with a single instruction. Bit-slice microprocessors are flexible devices, and their ALU size can be arbitrarily chosen by cascading an appropriate number of slices. This approach reduces decoding time considerably.

Second limitation, due to the sequential nature of these machines, was that only a single instruction could be executed at a time. Worst-case consideration of the decoding cycle reveals that such a system would support a low data rate[1]. System throughput can be increased considerably by the following strategy. A buffer memory may be used to store incoming code words during a decoding cycle. A realistic assumption can be made that

0141-9331/87/10527-07 $03.00

most of the received words are error free. This assumption holds in microwave and cable links, for example. Since in most cases the decoder requires very few cycles to process received words, the buffer memory would eventually be emptied. The only requirement is that the average incoming data rate should be lower than the average outgoing data rate. The decoder determines whether these code words are error free as they are received. An additional bit representing this information for each code word is stored in the buffer memory. At the beginning of each decoding cycle the decoder checks whether the received sequence is error free. If it is error free the decoder does not perform the decoding cycle. This checking takes very few clock cycles.

Below, the details of the hardware design are presented. For demonstration purposes, the microprogram for information set decoding (ISD)[2] is discussed. The ISD algorithm is a general example and can be applied to decode any linear block code. How to generate an (n, k) systemic linear binary block code and an explanation of the ISD algorithm can be found in Reference 2. There are several options available for implementing ISD algorithms. The parity check matrix H is used to determine a candidate error pattern and the collection of information sets is predetermined[2].

BIT-SLICE MICROPROCESSOR-BASED DECODER

This section explains the hardware details and operation of the decoder. A simplified block diagram of the system is shown in Figure 1. It has three main blocks:

- input port and error detector
- buffer memory manager (BMM)
- decoder and output port

Input port and error detector

A separate self-controlled input port and error-detection subsystem is incorporated in the front end of the system. This unit is synchronized with the received data, and is independent of the other units. It performs two operations:

- Receiving serial data and grouping it into blocks for decoding purposes.
- Determining whether the received sequence is error free.

Since both of these operations have to be performed on every block of a received sequence, a separate unit relieves the microprocessor from frequent interrupts to perform these tasks. The microprocessor is thus used solely for decoding purposes, increasing the system throughput. A block diagram of the subsystem is given in Figure 2.

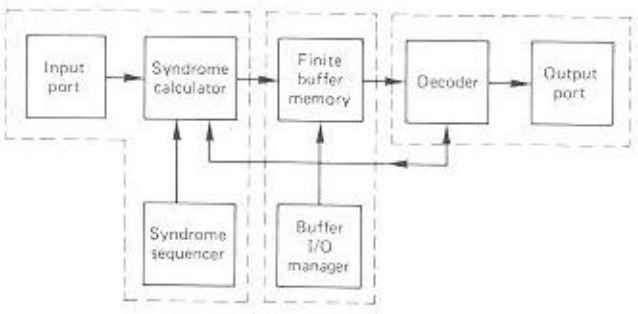

Figure 1. Simplified block diagram of the bit-slice microprocessor-based decoder

AR register. An incrementer register with clock and reset inputs. The contents are incremented on every clock pulse. The reset pin clears the register asynchronously. The output is used as the address field for the memory.

R_2 register. An $(n + 1)$ bit wide register. At each rising edge of the clock pulse the serial input is shifted into the most significant bit (MSB) of R_2. The MSB of this register is used as the load (LD) control for register R_1. The n least significant bits (LSBs) are connected to the n MSBs of R_{out}.

R_1 register. An $(n - k)$ bit wide register. This register stores the partial sums of the H^T matrix to calculate the syndrome as the data are being received. The input is the sum (modulo-2) of the data from the memory and its present contents. The output is also connected to an $(n - k)$ input OR gate. The RESET pin resets R_1 asynchronously.

R_{out} register. An $(n + 1)$ bit wide register. The n MSBs are taken from the n LSBs of R_2. The LSB is taken from the output of the OR gate. It has tristate outputs which are connected to the data lines of the buffer memory.

EPROM H. An $n \times (n - k)$ memory. The H matrix is arranged by columns. The syndrome is calculated by adding the appropriate columns of the H matrix as the data are being received.

WR flipflop. A logic 1 output from this flipflop indicates that a new block has been loaded in R_{out} and is ready to be read by the buffer memory manager (BMM).

The operation of the circuit is as follows. Assume that prior to the first rising edge of the clock the register R_1 and the counter register AR have been cleared. The MSB of R_2 has been loaded with the first received bit. The data lines of the memory now have the first row of H^T. At each of the next successive n rising edges of the clock the following changes occur.

- If the MSB of R_2 is a 1, the data on DBUS is summed with the contents of R_1 and the new sum is loaded in R_1.
- The AR register/counter is incremented.
- R_2 is shifted right with the SI input entering the MSB of R_2.

Thus after n rising edges the register R_1 will have the syndrome. The n LSBs of R_2 have the received sequence. The MSB of R_2 has the first bit of the next block. A flag is available at the output of the OR gate, and a 1 indicates that the received sequence is a noise corrupted code word. The following operations take place at the nth falling edge.

- The received sequence is loaded along with the flag in R_{out}.
- The AR and R_1 registers are cleared.
- The WR flipflop is set.

Before the $(n + 1)$th rising edge the situation is the same as before the first rising edge and the cycle repeats.

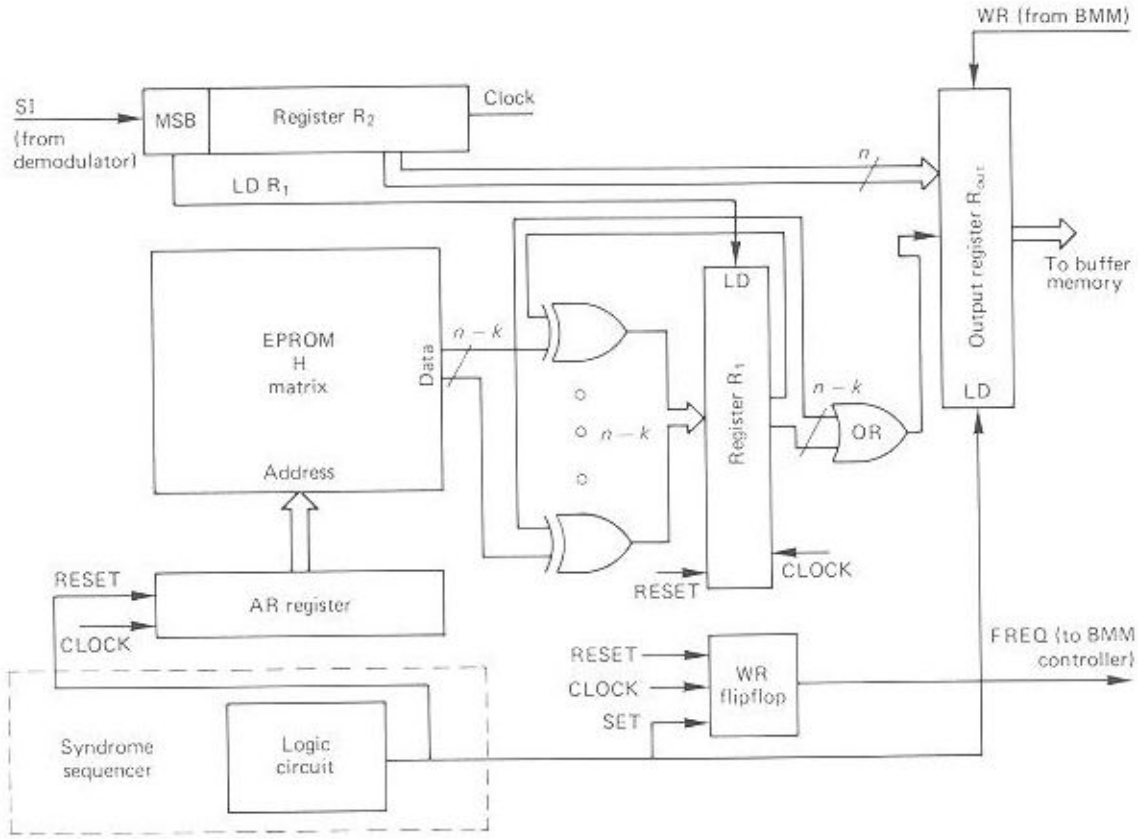

Figure 2. Block diagram of the input port and error-detection circuit

Buffer memory manager

During a decoding cycle several blocks may be received depending on the data rate λ. These blocks have to be stored in successive memory locations in the buffer memory (BM). Furthermore, the decoder should be able to read blocks from correct memory locations in order. These functions are performed by the BMM. A simplified block diagram of the BMM controller is shown in Figure 3. Its inputs and outputs are described below.

Inputs

EQ: high indicates that the contents of counters EP and FP are equal (see Figure 4).

FREQ: high indicates that a block in R_{out} is ready to be stored in the buffer memory.

EREQ: high indicates that the decoder is ready to read data from BM.

F: high indicates that the buffer is full.

E: high indicates that the buffer is empty.

Outputs

DR: goes to the CC (condition code) multiplexer of the decoder. Low indicates that the data are ready on the buffer memory data (BMD) bus.

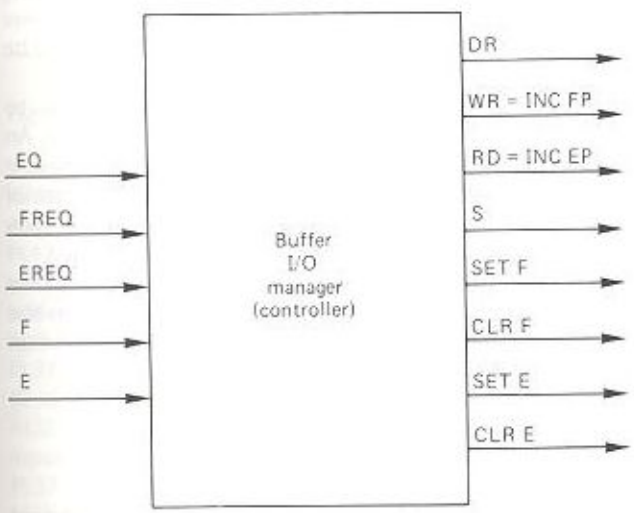

Figure 3. Block diagram showing the buffer memory manager controller

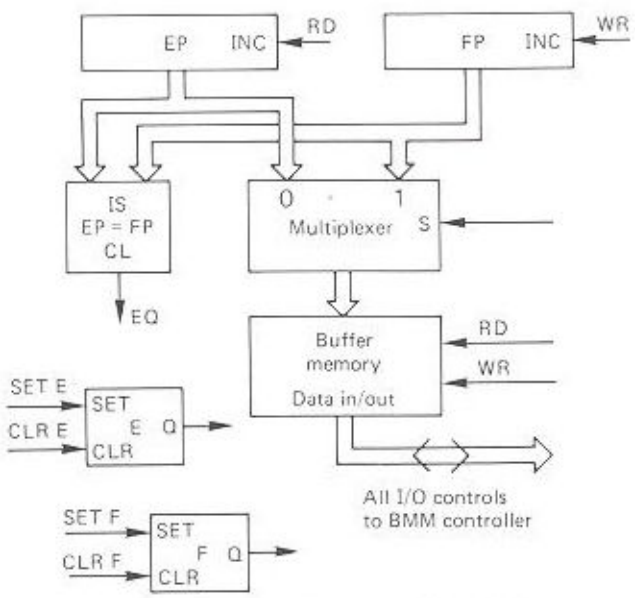

Figure 4. Simplified block diagram of the buffer memory manager

WR=INC FP: enables BM to input data and increments the FP counter.
RD=INC EP: enables BM to output data and increments the EP counter.
S: selects either of the two sources of address to BM: 0 selects EP and 1 selects FP.
SET F (E): 1 sets F (E) flipflop.
CLR F (E): 1 clears F (E) flipflop.

Building blocks

EP and FP are mod M clocked counters, where M is the capacity of the buffer memory to hold $(n + 1)$ bit received sequences. The contents of EP and FP point to the location where the next block is to be read from BM, and where the next block is to be stored in BM, respectively. There are m multiplexers (2×1), where m is the number of bits in EP or FP. CL is the combinational logic which determines if the contents of EP and FP are equal. BM stores incoming data. E and F are flipflops with clocked SET and RESET inputs. A high in E indicates that BM is empty. A high in F indicates that BM is full.

Sequence of operation

The flowchart for the operation of the BMM is shown in Figure 5. FREQ is given priority and is checked first, then EREQ from the decoder. In response to the FREQ high signal the FP pointer is selected as the address source. WR and INC FP signals are made high if the F flipflop is not high, indicating that BM is not full. The memory is full if after writing these data the FP pointer is the same as the EP pointer. The EP pointer points to the location where the data are still to be read by the decoder. In response to the EREQ high signal the EP pointer is selected as the address source. RD=INC EP (Figure 3) is made high if the E flipflop is not high, indicating that BM is not empty. The memory is empty, i.e. the E flipflop is set, if after reading this data the EP pointer is the same as the FP pointer.

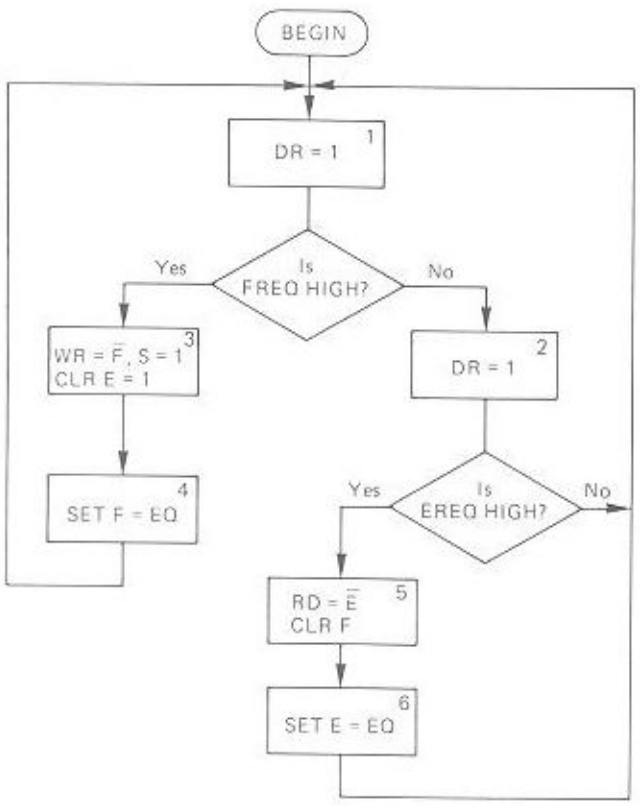

Figure 5. Flowchart for the operation of the buffer memory manager controller

Decoder design

This section explains the design and the operation of the bit-slice microprocessor-based decoder. For detailed information on these devices refer to References 4 and 5.

Figure 6 shows the block diagram of the design. The left half of the figure is the microprogram sequencer and the right half is predominantly the data handling portion. The microprogram sequencer essentially consists of an instruction register, controller, microprogram memory, pipeline register and CC (condition code) multiplexer. The instruction register is loaded with the starting address of the microprogram sequence at power-up or when changing decoding algorithms. These address lines are connected to the D_i inputs of the Am2910 controller.

The Am2910 selects the next microprogram instruction to be executed. Instructions are taken from the pipeline register. The CC multiplexer selects the appropriate condition inputs determined by a field from the pipeline register. The Y_i outputs of the Am2910 have the next address which is applied to the address pins of the microprogram memory. The microinstruction appears at the data lines of the memory and is loaded in the pipeline register at the next rising edge of the clock.

The data handling portion of the design consists of the Am2901 ALU, MAR, H storage memory, MO register, status register, output register, parity detect combinational logic and control circuitry for shift operations. Four slices are cascaded in this design, sufficient for operating on 16-bit words. The design incorporates two buses: Y bus and data bus.

Y bus is connected to the Y data output lines of the Am2901. It normally carries the F output of the ALU. This bus is connected to the MAR inputs for the initial address transfer. A parity detect combinational logic connected to this bus determines the parity of the bus signals. The parity flag can be loaded in the status register. The output register is also connected to the Y bus. The data bus is connected to the D inputs of the ALU. The source can either be selected from the MO register or data from BM. One of these sources can be selected by enabling appropriate tristate buffers. Two buffers and one inverter are connected to form shift control logic. A 0 or a 1 can be transferred into the MSB or the LSB of the ALU.

During the decoding process the decoder has to continually read successive locations in the memory. An added feature of incrementing/decrementing is therefore included in the MAR to relieve the ALU from the task of incrementing or decrementing the MAR, as the case may be.

A data register MO, connected to the data output lines of the memory, breaks the critical delay path from the memory to the ALU, thereby imposing less restriction on the speed of the memory.

MICROPROGRAM STRUCTURE

The microprogram instruction format for the design shown in Figure 6 is 48 bit wide. The function of the microprogram control bits are described below.

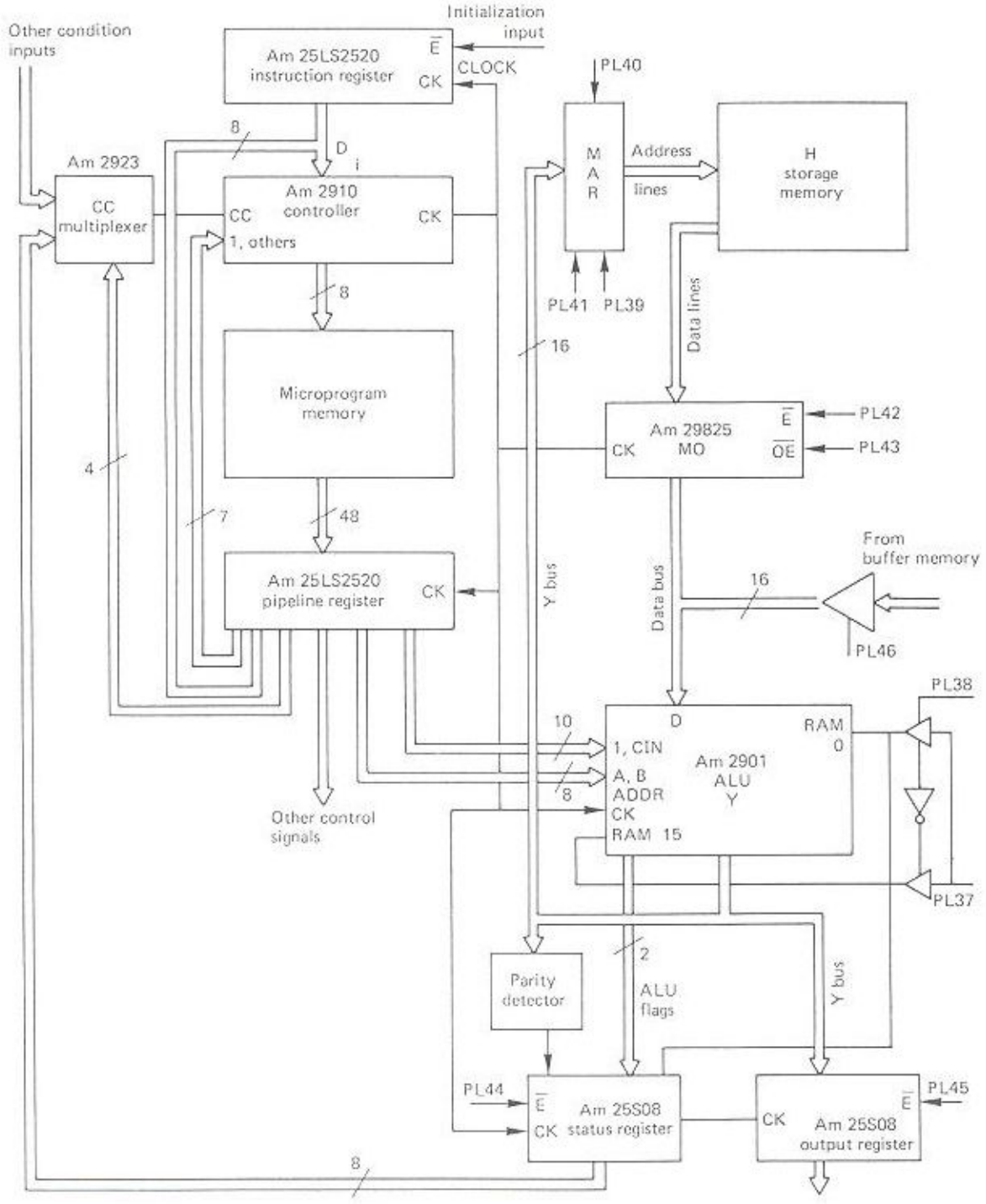

Figure 6. Simplified block diagram of the decoder

An 8 bit wide field PL0–PL7 usually serves as the next microprogram address and also as an initial count in the Am2910. PL8–PL11 are the I input of the Am2910 sequencer. PL12–PL14 are RLD and CCEN control inputs and C_i input to the Am2910. PL15–PL18 select one of the condition codes to be tested. PL19–PL22 are used for the A address and PL23–PL26 for the B address of the Am2901 ALU. PL27–PL29, PL30–PL32 and PL33–PL35 are for ALU destination control, for ALU function control and for ALU source operand control, respectively. PL36 is the C_i input to the least significant Am2901 slice in the ALU. PL37 is the bit to be shifted into the MSB or LSB of the ALU (depending on PL38). PL39–PL41 serve input to LD, CET and U/$\overline{D}$ control inputs, respectively, of the binary up/down counter (Am25LS2569) acting as MAR to the H storage memory. PL47 serves to send a request for a new block to BMM.

Information set decoding algorithm

The microprogram for the algorithm is given in Figure 7. To execute the algorithm for a particular code, the (15, 5) code in this case, covering patterns and corresponding parity check matrices are calculated initially. These parity check matrices are stored in the H memory by rows. A mask for the particular covering pattern follows each matrix. Eight parity check matrices, each of them followed by the mask, are stored in the memory. Each step (micro-instruction) is described below.

MEM LOC	Bra Add	2910 IRC C LCCO DEIN N D M U X	2901 ABIIICSS NHH AA852 DD--- BU DD630 I/ T- D	MAR MCM AEA RTR L U D / D	MEM EO E	SO TU AT T RR EE GG	EE NM P BT UY F MR EE MQ
1	xxx	E111x	xx144xxx	01x	1x	x1	00
2	xxx	E111x	xx1xxxxx	101	0x	x1	00
3	xxx	E111x	x4367xxx	101	00	x1	00
4	xxx	E111x	x6367xxx	101	00	x1	00
5	xxx	E111x	x7367xxx	101	00	x1	00
6	xxx	E111x	x6133xxx	01x	1x	x1	00
7	xxx	E111x	45334xxx	101	0x	x1	00
8	008	31019	xx1xxxxx	11x	1x	x1	00
9	009	3101A	xF367xxx	11x	11	x1	11
10	xxx	E111x	xF533x00	11x	11	01	00
11	***	3101C	xx1xxxxx	11x	1x	x1	00
12	009	4101F	xB344xxx	11x	1x	x1	00
13	xxx	E111x	FA345xxx	101	00	01	00
14	016	3101D	xx1xxxxx	11x	1x	x1	00
15	017	3101F	xB733x11	11x	1x	x1	00
16	xxx	E111x	xB733x01	11x	1x	x1	00
17	xxx	8111x	BA333xxx	11x	1x	x1	00
18	009	4101F	xB333xxx	11x	1x	x1	00
19	xxx	E111x	xA533x00	11x	1x	01	00
20	022	3101C	xx1xxxxx	11x	1x	x1	00
21	xxx	E111x	x83031xx	11x	1x	x1	00
22	xxx	8111x	xx1xxxxx	11x	1x	x1	00
23	xxx	E111x	783211xx	11x	1x	01	00
24	x30	3101E	xx1xxxxx	11x	1x	x1	00
25	xxx	E111x	xx1xxxxx	101	00	x1	00
26	xxx	E111x	x53130xx	11x	1x	01	00
27	012	31018	xx1xxxxx	11x	1x	x1	00
28	xxx	E111x	xF733111	11x	1x	x1	00
29	006	3101F	xF133xxx	11x	1x	x0	00
30	xxx	E111x	x9344xxx	11x	1x	x1	00
31	009	4101F	x8367xxx	100	1x	x1	00
32	xxx	E111x	xB533x00	100	1x	01	00
33	035	3101C	xx1xxxxx	11x	0x	x1	00
34	xxx	E111x	99365xxx	11x	10	x1	00
35	xxx	8111x	89341xxx	11x	1x	x1	00
36	xxx	E111x	9F361xxx	11x	1x	x1	00
37	xxx	E111x	xF733x01	11x	1x	x1	00
38	006	3101F	xF133xxx	11x	1x	x0	00

Figure 7. Microprogram for information set decoding

Steps 1–5: Data stored in the first three locations of the memory are loaded into registers R_4, R_6 and R_7, respectively. Data stored in the memory are the number of covering patterns, the starting address for the first parity check matrix and the error-correcting capability of the code, respectively. These states are entered once only, before the execution of the algorithm.

Steps 6 and 7: The MAR is initialized with the starting address of the first parity check matrix. At the end of these states, MO will have the first row of the H matrix and MAR will be addressing the second. A copy of R_4 is made in R_5.

Steps 8 and 9: Check and wait until BM is not empty and then read the received sequence in R_5.

Steps 10 and 11: Determine whether the received sequence is erroneous or not. If error-free go to state 37; else, continue.

Step 12: Initialize the controller to execute the next loop ten times; also initialize R_{11}.

Steps 13–17: Calculate a syndrome of the received sequence by executing this loop ten times. Each time a syndrome is computed a different parity check matrix is used. At the end of this loop the syndrome will be in R_{10} and R_{11}.

Step 18: Initialize the controller to execute the next loop ten times.

Steps 19–22: Calculate the weight of the syndrome by executing this loop ten times. R_8 will contain the weight at the end of this loop.

Steps 23 and 24: Check whether the weight of the syndrome is less than or equal to the error-correcting capability of the code. If yes, go to state 30; else, continue.

Steps 25–27: Increment MAR to point to the first row of the next parity check matrix. Determine whether all the parity sets have been exhausted. If yes, continue; else, go to state 12.

Steps 28 and 29: Output the received sequence with an error flag in the LSB of the OR register, and go to state 6.

Steps 30 and 31: Fetch the mask for the present parity check matrix in R_8. Initialize the controller to execute the next loop ten times.

Steps 32–35: Calculate the error pattern by adding the appropriate rows of the H matrix and ANDing with the mask. This loop is executed ten times at the end of which R_9 contains the error pattern.

Step 36: Calculate the code word by summing the received sequence with the error pattern.

Steps 37 and 38: Output the code word with no-error flag and go to state 6.

DECODING TIMES AND MEMORY REQUIREMENT

The total time (in seconds) to decode at the ith information set can be calculated and shown to be[6]

$$\tau_i = [11 + i[7 + 7.5(n-k)] + 3.5(n-k)] \times 10^{-7}$$

The memory required to store the parity check matrices is given by $n(k+1)N_{cov}$, where N_{cov} is the number of covering patterns for the (n, k) code. For a (15, 5) triple-error-correcting code a memory of 640 bit is needed, and for a (48, 24) five-error-correcting code ~108 kbit needs to be stored. The buffer memory requirements have been studied against several system parameters. The solution to the mathematical queueing model of the buffer and the results of the study are available in Reference 6. The results show that the buffer memory requirements are modest even for high data rates and high bit-error probability. For example, with a data arrival rate of 960 kbit s^{-1} and a bit error probability of 10^{-4}, for a (48, 24) quadratic residue code with the information set decoding algorithm, the required buffer size was found to be 12.3 kbit with an overflow probability of less than 10^{-20}. If the data rate decreases, the required buffer size decreases.

CONCLUSIONS

A bit-slice microprocessor-based architecture to decode linear block codes has been described. The error-detection and buffer memory management functions are assigned to front-end hardware logic. The resulting improvement in performance is considerable for relatively simple hardware additions. Study of the buffer memory requirements shows that the requirements are modest and do not therefore involve a major hardware addition. For smaller block lengths to be handled by the decoder, parameter changes in the microprogram and syndrome sequencer are required. The same design concept can be used to implement decoders for larger codes.

ACKNOWLEDGEMENT

The authors wish to acknowledge King Fahd University of Petroleum and Minerals, Dahran, Saudi Arabia, for support of this research.

REFERENCES

1 **Said, S M and Dimond, K R** 'Realtime implementation of the Viterbi decoding algorithm on a high-performance microprocessor' *Microprocessors Microsyst.* Vol 10 No 1 (January/February 1986) pp 11–16

2 **Clark, G C and Cain, J B** *Error-correction and coding for digital communications* Plenum Press, New York, NY, USA (1981)

3 **Prange, E** 'The use of information sets in decoding cyclic codes' *IRE Trans. Inf. Theory* Vol IT-8 (1962)

4 **Mick, J and Brick, J** *Bit-slice microprocessor design* McGraw Hill, New York, NY, USA (1980)

5 *Bipolar microprocessor logic and interface data book* Advanced Micro Devices, Sunnyvale, CA, USA (1983)

6 **Soomro, A A** 'A hardware-software approach to implement decoding algorithms' *MS thesis, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia* (January 1987)

Amjad A Soomro received BS and MS degrees in electrical engineering from King Fahd University of Petroleum and Minerals in 1984 and 1987 respectively. He is presently working at Dammam Computer Center as a computer engineer, in the area of computer networks in banking environments. His interests are computer networks and data communications.

Dr Sadiq M Sait received his BE degree in 1981 from the University Visvesvaraya College of Engineering, Bangalore, India, and his MS and PhD degrees in electrical engineering from the King Fahd University of Petroleum and Minerals in 1983 and 1986, respectively. He is currently an assistant professor at KFUPM. His research interests are in hardware description languages and VLSI design automation.

Dr Mushfiqur Rahman received his BSc in electrical engineering from University of Engineering and Technology, Dhaka, Bangladesh, in 1969. He received MASc and PhD degrees in electrical engineering from the University of Waterloo, Canada, in 1972 and 1975 respectively. During 1975–1976 he worked at Instituto de Pesquisas Espacias, Sao Paulo, Brazil. Since 1976 he has been with the King Fahd University of Petroleum and Minerals, where he is currently an associate professor. His areas of interests are coding, communications and computer networks.