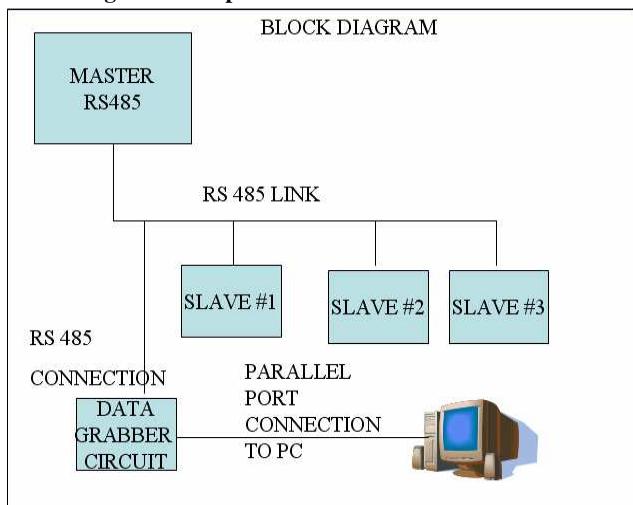


DATA GRABBER CIRCUIT

AUTHOR : MOIZ HUSAIN HAIDRY
B.E FINAL YEAR
DISCIPLINE: ELECTRONICS & INSTRUMENTATION
INSTITUTE OF ENGINEERING AND TECHNOLOGY,
DEVI AHILYA UNIVERSITY

ABSTRACT- This paper presents the aim and development of the Data Grabber Circuit which I had designed during my research internship in the Raja Ramanna Centre for advanced technology, Department of nuclear Energy, Govt. of India. The aim of the circuit was to continuously capture data from serial communication over RS 485 Master-Slave Link. As data transmission in serial communication is done at 750kbaud/sec it cannot be directly connected to a computer because of the incompatibility in baud rate. Hence I made a circuitry using a microcontroller that would synchronize incoming serial burst data with the computer processing by buffering the incoming data, converting the serial data into parallel one and transmitting the same using parallel port interfacing to the computer.



The incoming serial data is stored in a 32K RAM and the PC reads this data from the RAM using its parallel port. The process of reading the data from the RAM involves a sequence of handshaking signals between the PC and microcontroller to ensure that there is no data loss on the part of the data reception from the PC. Also the data is transmitted from the microcontroller to the parallel port via a latch to ensure the same. Further the process of reading and writing data in the RAM follows the circular queue algorithm so that optimum usage of the RAM is done and conditions of overflow and underflow can be easily detected. This circuit is made for storing bursts of incoming data and not to be used for continuous flow of data due to chances of an overflow. The circuit makes use of Dallas's DS89C420 microcontroller (which follows 8051 assembly code). Also the circuit is compatible with an RS232 link. The assembly code is made using the basic 8051 assembly language. The interfacing code has been designed in both TurboC and Visual C++ environment.

I DESIGNING:

- First of all as it is required to transfer data being obtained by microcontroller to the computer we have two options-
 - 1) Parallel Transmission
 - 2) Serial Transmission

As we need to save more and more time parallel transmission is the obvious choice as the processing and transfer time is least. This can be achieved by interfacing the microcontroller circuit with the parallel port (or printer port of the computer) and using this port in bi-directional byte mode. Also we can do handshaking between the microcontroller circuit and the computer by making use of the output line of the printer port's control register and the input lines of the printer port's status register. The data shall be transferred using the printer port's data lines which shall be connected to a 74LS374 latch and in parallel with a 74LS244 buffer. This shall be used to make the port flexible for bi-directional data transfer. The output lines are latched and the input lines are buffered and different logic levels of the same handshaking line shall control their respective enable pins such that the port is an output port by default. However in this design we shall focus only on parallel port data reception and hence shall not attach the 74LS244 buffer in parallel.

- Second requirement is of a buffer, which shall save the incoming bursts of serial data, and this data is then transferred to the parallel port via a latch. Hence I interfaced a 32K RAM with the microcontroller which is a DS89C420. The multiplexed address and data lines of PORT0 of the Microcontroller was demultiplexed using 74LS373 Latch and hence I obtained the data lines D0 to D7 directly from PORT0 and the address lines A0 to A7 from the Latch output which is controlled by the Address Latch Enable (ALE) pin of the Microcontroller. The remaining address line A8 to A14 was obtained by the lines P2.0 to P2.6 of PORT2 of Microcontroller and the remaining line P2.7 acted as the Chip Enable(nCE) of the RAM. The Read(nRD) line was connected to the output of an AND gate whose inputs were P3.7(nRD) and PSEN(Program Store Enable) lines of the

Microcontroller so that the RAM can be used to store both data and code.

- Next I connected the serial lines of the DS89C420 to the respective line drivers. I have interfaced the circuit with two standard serial ports
1)RS485 – To interface with RS485 serial port I connected the serial pins of the Microcontroller to the line driver 75176. The TXD0 and RXD0 lines of the Microcontroller was connected to the D and R pins of the line driver respectively via jumpers and the pin P1.7 was connected as a control line to the nDE and RE pins of 75176. The outputs A and nB of 75176 was connected to the 3 and 8 pin of a DB9 male connector and pin 5 is connected to Ground according to the RS485 serial port configuration. The outputs were also connected to 390 ohms,220 ohms,and 390 ohms biasing resistors via jumpers to be connected according to requirements.
2)RS232 - To interface with RS232 serial port we connected the serial pins of the Microcontroller to the line driver MAX232.I had

an optional connection of both TXD0,RXD0 and TXD1,RXD1 lines of the Microcontroller to the T1IN and R1OUT lines of the MAX232 with help of jumpers. Also I connected four 10uF capacitors as is required in MAX232.The output lines T1OUT and R1IN of line driver IC was connected to pin 2 and pin 3 of a DB9 male connector respectively and the pin 5 was connected to Ground according to RS232 serial port configuration.

Please not that my focus was to copy bursts of serial incoming data and transmit it continuously to the computer's parallel port, hence it is not required for the parallel port to process as fast as the serial port until the buffer is not overflowed which is difficult to achieve as the buffer (RAM) is of large size. Therefore I had room to achieve complete handshaking.

The following is a schematic design of the circuit:-

II COMPONENTS

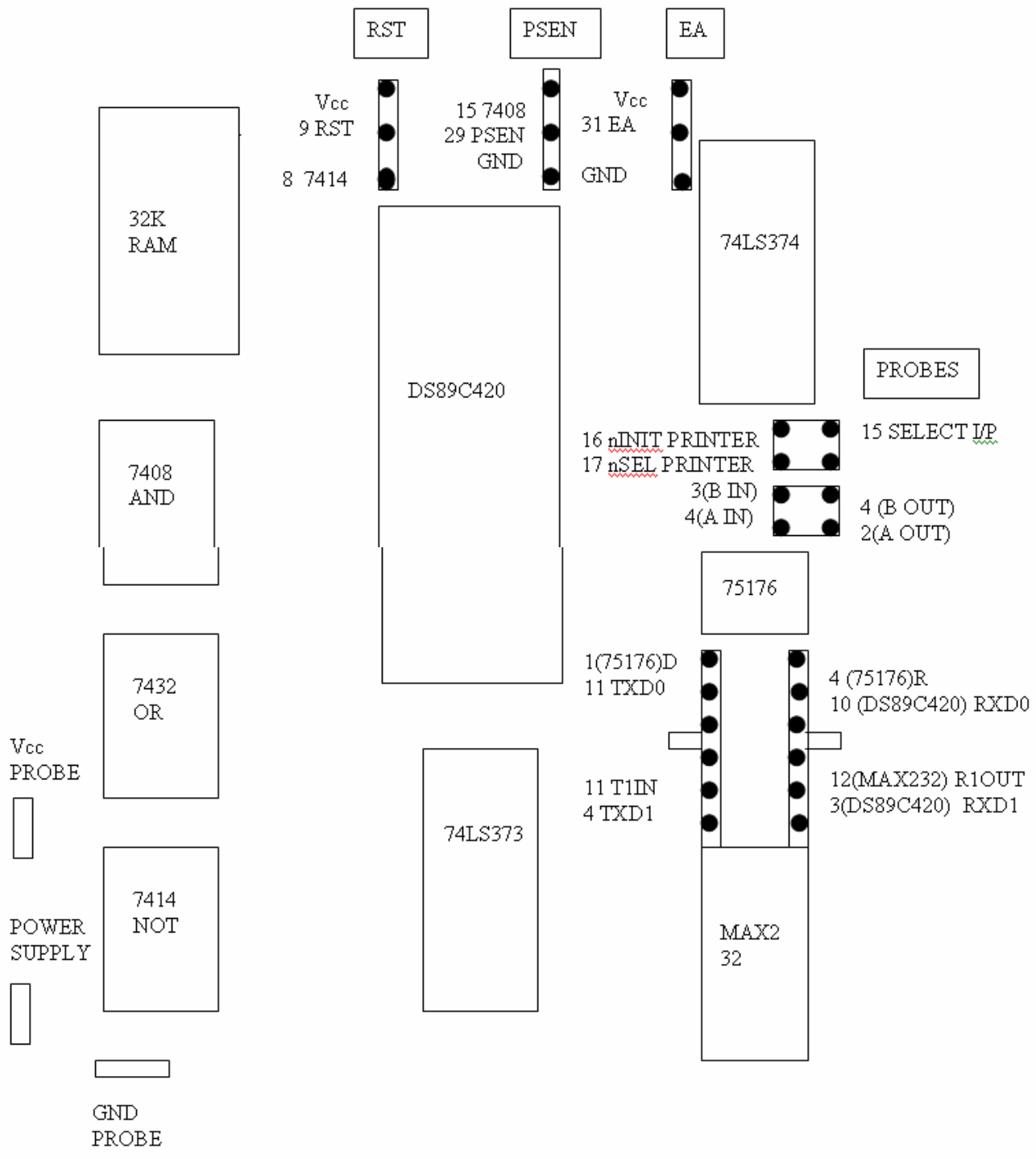
The following components were studied and implemented to develop the circuit:-

- RS485 Standard Serial Port
- RS232 Standard Serial Port
- 75176 line driver IC
- DS89C420 microcontroller
- 32K RAM
- 74LS374 Latch
- 74LS373 Latch
- MAX232 line driver IC
- 74LS244 Buffer
- 7414 NOT logic gate
- 7408 AND logic gate
- 7432 OR logic gate

III DEVELOPMENT OF PROTOTYPE

The circuit schematic was designed using PROTEL schematic designer and the corresponding circuit been made by soldering the various components and wiring the connections between the corresponding pins on a zero PCB.

Here is an outline of the circuit-



IV CONNECTION OF THE CIRCUIT TO THE HYPER TERMINAL

The circuit was set in the programming mode where the program was to be stored in the external ram. The following jumper settings were done in the circuit.

* The RESET pin was connected to push-button circuit for PUSH BUTTON RESET MODE

* The PSEN pin was connected to 7408 AND LOGIC GATE as an input with the READ PIN(P3.7) as second input

* The EA pin is connected to Vcc.

* The TXD1 and RXD1 pins of the DS89C420 microcontroller was connected to the T1IN and R1OUT pins of MAX232 respectively.

* The T1OUT and R1IN pins of the MAX232 IC was connected to the 3 and 2 pins of a DB9 male connector respectively.

When the circuit with the above settings was connected to the serial port of the computer a prompt was obtained in the hyperterminal. The RESET button was tested and it worked properly.

V PARALLEL PORT INTERFACING WITH THE CIRCUIT

Assembly programs were loaded into the external RAM via the hyperterminal and these programs were used to test the parallel port interfacing. In the interfacing between the computer and the circuit, the following handshaking occurs before the data transfer-

- The Microcontroller first sets the pin (P1.6) when data is available in the RAM to be read. This line is connected to the nERROR, a status line of the parallel port. By this the Microcontroller indicates that data is available to be read from the latch.
- The computer then generates an interrupt signal, when it is ready to read, by the help of the nSELECT PRINTER (Pin 17), a control pin of the parallel port, which is connected to the External Interrupt pin INTO (Pin 12) of Microcontroller. This Interrupt is set to be Edge Triggered.
- The 74LS374 latch's clock pin CLK is connected to the output of an OR LOGIC GATE 7432 whose inputs are the WRITE pin(P3.6) and nP2.7 which defines the latch's address. The Output Enable pin (OE) is connected to the nINITIALIZE PRINTER (Pin 16) and is used to output the data from the latch to the parallel port.

The latch is enabled when the the Write signal is generated and the P2.7 line is high. Accordingly the address of this latch is defined to be 8000H.

VI TRANSMISSION OF DATA FROM RAM TO PARALLEL PORT STRUCTURE OF PROGRAM

The program is basically divided into two parts-

- In one part a mathematical operation such as increment or rotation was performed upon the contents of the accumulator to generate expected pseudo data and that data is stored in the externally connected RAM(Random Access

Memory). This part of the program is executed in the main routine of the code.

- The other part is executed as the interrupt subroutine. In this part the of the code the data is obtained from the RAM in the accumulator and later this data is transferred to the latch. This data is then outputted from the latch to the parallel port.

The address of both the RAM and the latch are defined and are explained later. There are two data pointer registers DPTR and DPTR1 in the DS89C420. DPTR1 is accessed by setting the first bit of the DPS register. The DPTR register is used to refer to the RAM location where is to be written and DPTR1 is used to refer to the location from where data is to read.

VII IMPLEMENTATION OF THE CODE AS A CIRCULAR QUE

In the program above there is a chance that values would be overwritten upon the RAM before they are read by the parallel port and hence there may be a data loss due lack of synchronization. To sort this problem out I shall implement the above code as a circular que and hence apply more checks and compare the values of the two data pointers. First of all we shall see the algorithm of a circular que and later how it is implemented in assembly language-

A. CIRCULAR QUE AND ITS ALGORITHM

When a new item is inserted at the rear, the rear moves upwards. Similarly, when an item is deleted from the queue the front arrow moves downwards. After a few insert and delete operations the rear might reach the end of the queue and no more items can be inserted although the items from the front of the queue have been deleted and there is space in the queue. To solve this problem, queues implement wrapping around. Such queues are called Circular Queues. Both the front and the rear wrap around to the beginning of the array when they reached the MAX size of the queue. It is also called as "Ring buffer".

B. CIRCULAR QUEUE INSERTION

QINSERT (QUEUE, N, FRONT, ITEM)

This procedure insert an element ITEM into a queue.

1. [Queue already filled?]

IF (FRONT==1 and REAR==N) or FRONT ==REAR + 1,then:

write: overflow, and Return

2.[Find new value of REAR]

IF FRONT==NULL then [Queue initially empty.]

Set FRONT=1 and REAR=1

ELSE IF REAR ==N then

Set REAR=1

ELSE

set REAR=REAR+1

[End of if structure]

3. Set QUEUE[REAR]=ITEM.[This inserts new element]

4. Return.

C. CIRCULAR QUEUE DELETION

QDELETE(QUEUE, N, FRONT, REAR, ITEM)

This procedure deletes an element from a queue and assigns it to the variable ITEM

- 1.[Queue already empty]
- if FRONT=NULL then write UNDERFLOW, and Return
2. Set ITEM=QUEUE[FRONT]
3. [Find new value of FRONT]

If FRONT =REAR then [Queue has only one element to start]

Set FRONT=NULL and REAR=NULL

Else if FRONT ==N then

Set FRONT=1

Else

Set FRONT=FRONT +1

[End of if statement]

4.Return

VIII THE FINAL PROGRAM AND ITS EXPLANATION

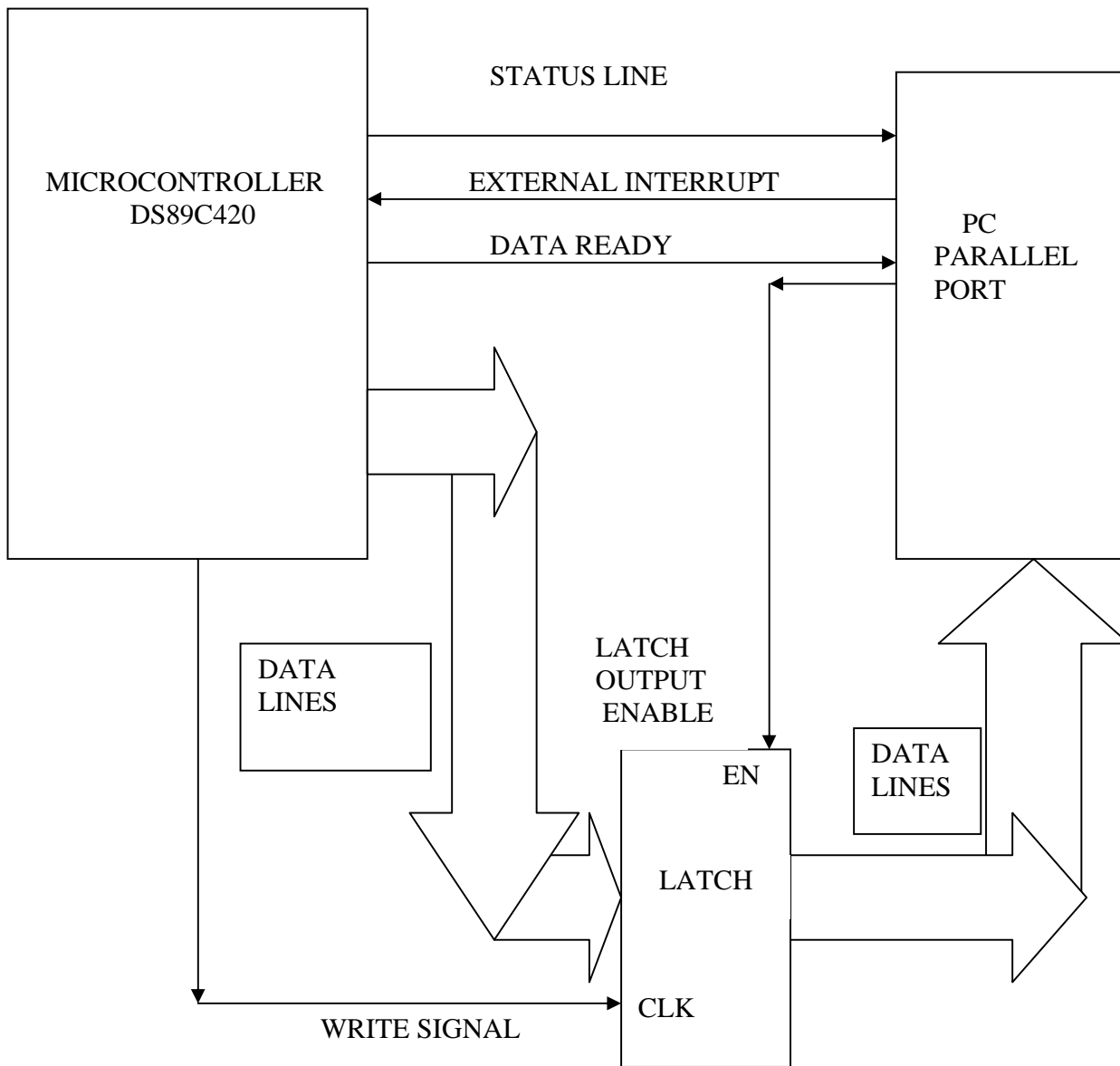
A. OVERVIEW-

We recall that our program is expected to perform the following tasks and how is it implemented for testing-

- First we have to keep on fetching data from the serial port and save it into a RAM(buffer).

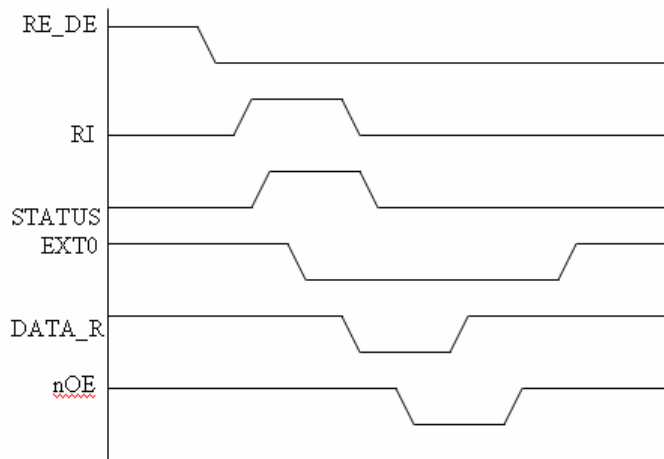
- Then we have to keep on transmitting data from the buffer to the computer's parallel port.
- We need to implement the buffer as a circular buffer ,by implementing a circular queue, algorithm so that there is no sort of data loss and if there is overflow due to loss of synchronization it shall be signal and intake of data shall be stopped.
- The process of fetching data from the serial port and saving it into the buffer is implemented as a circular queue insertion function. In the program it shall be a serial interrupt subroutine.
- The process of placing data from the buffer to the parallel port is implemented as a circular queue deletion function. In the program it shall be a external interrupt subroutine.
- The external interrupt has higher priority than the serial interrupt however we expect the program to keep on loading data from the serial port into the buffer hence we need to assign higher priority to the serial interrupt. This is done using the PE register.
- Various Handshaking signals are used to achieve proper synchronization, these signals are explained in the next section.

**B. HANDSHAKING
BLOCK DIAGRAM**



C. CONNECTION OF VARIOUS LINES

STATUS FLAG – FROM P1.6 OF
MICROCONTROLLER TO PARALLEL PORT PIN NO.
13 (nERROR)
EXT. INTERRUPT-FROM PIN NO.17 (nSELECT
PRINTER) OF PARALLEL PORT TO INT0(P3.2) OF
MICROCONTROLLER
DATA READY FLAG-FROM P1.5 OF
MICROCONTROLLER TO PARALLEL PORT PIN
NO.13(SELECT INPUT).
LATCH OUTPUT ENABLE - FROM PIN NO. 16(nINIT
PRINTER) TO OUTPUT ENABLE (nOE) OF LATCH



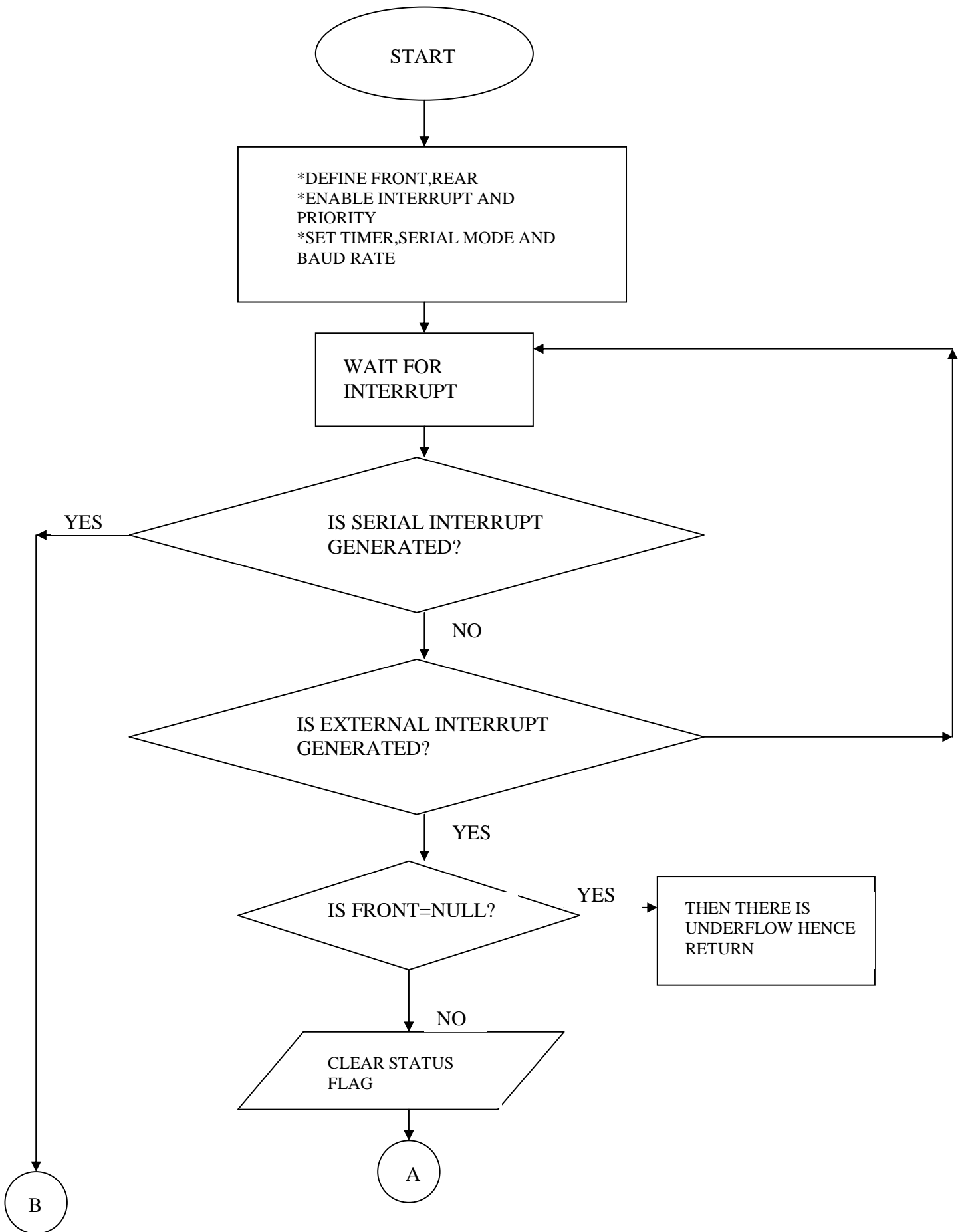
- First of all RE_DE the Active low Receive Enable of 75176(Driver IC for RS485) is activated at the start of the program.
- After that the microcontroller receives the incoming data and activates the RI flag.
- The serial interrupt subroutine is then called the data is written into the RAM and then the STATUS flag is activated to indicate that data is available.
- After that the program in the computer, which was waiting for the STATUS flag, proceeds to

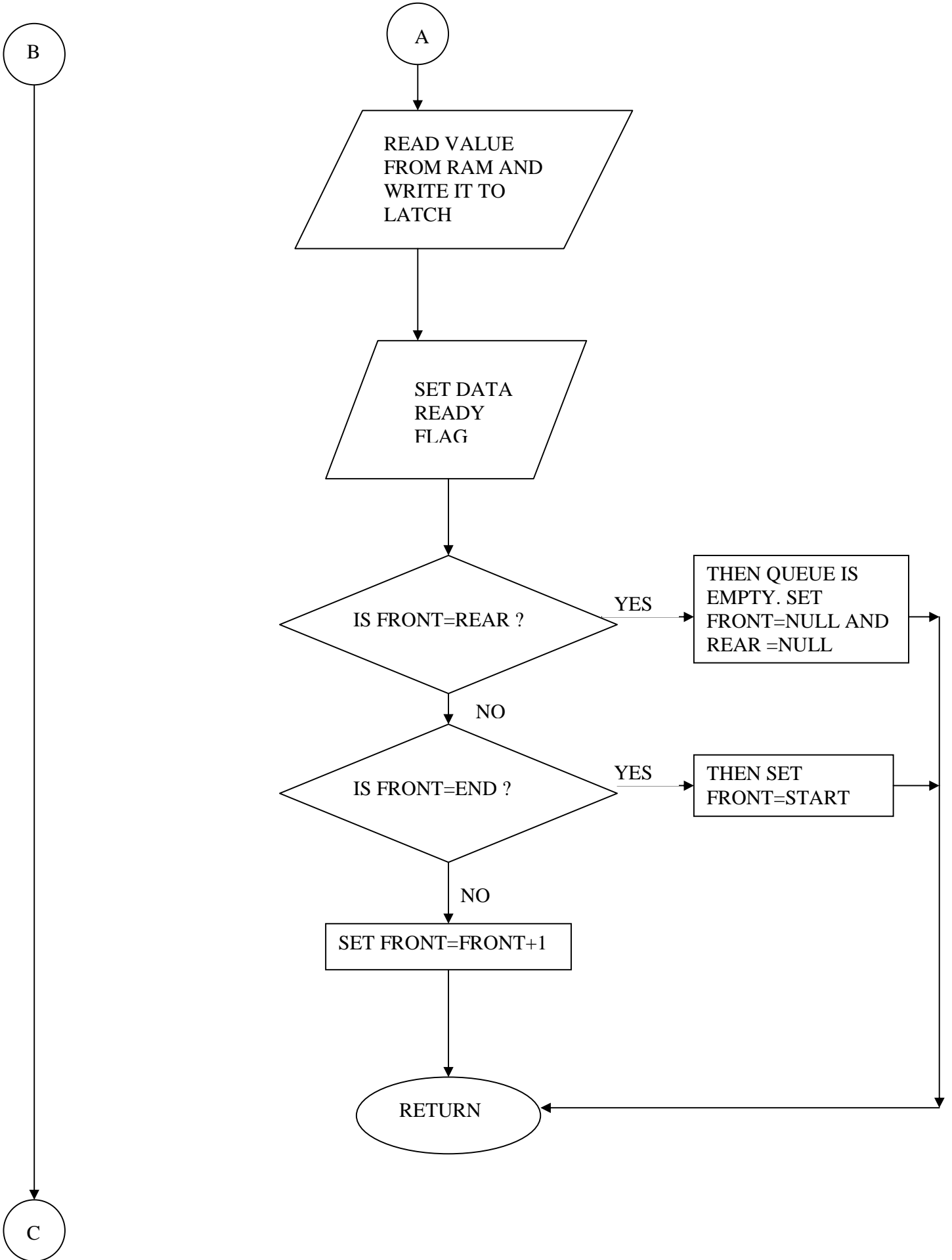
generate an external interrupt by generating an edge triggering signal EXT0.

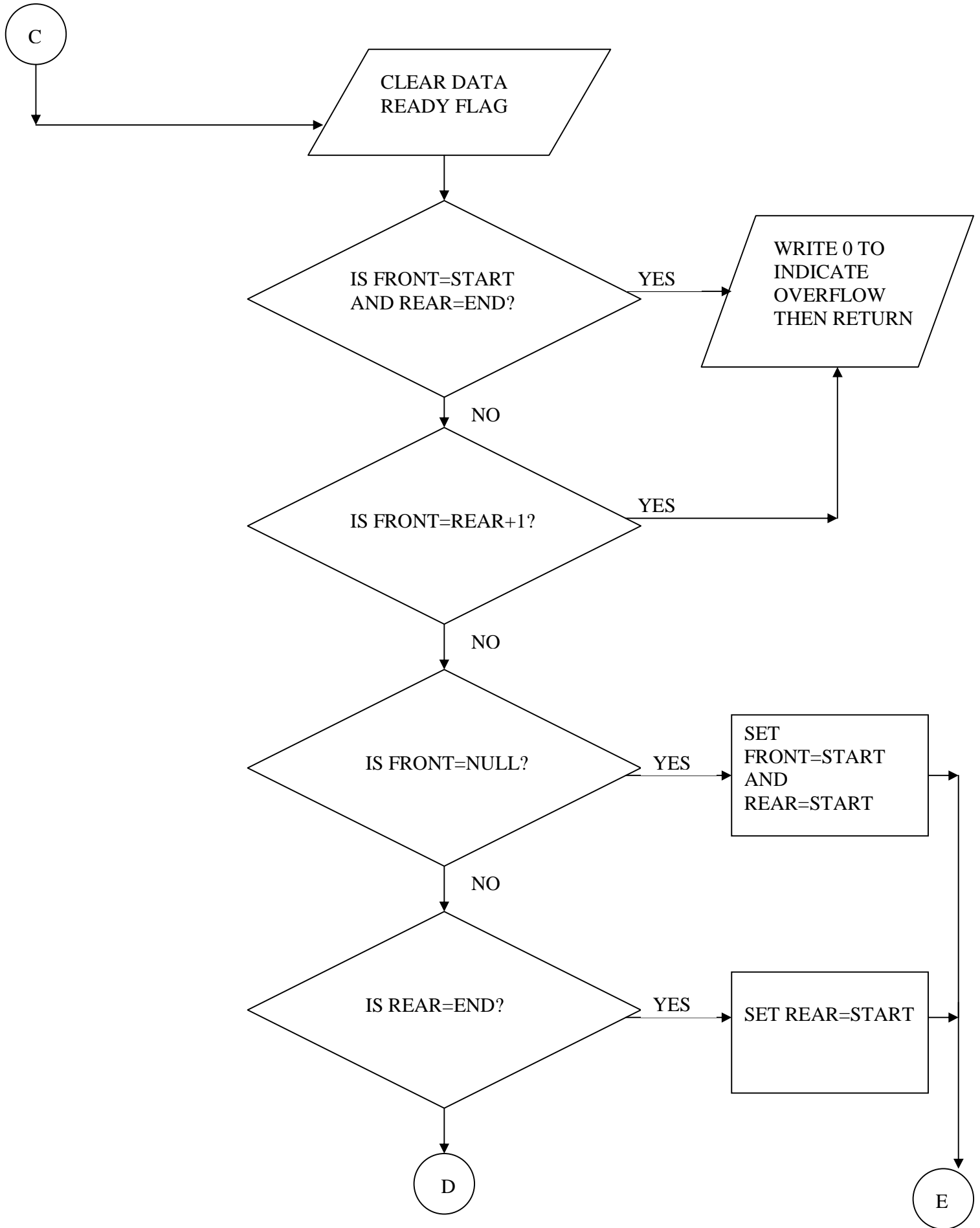
- The EXT0 signal calls the external interrupt subroutine, where the data is placed from the RAM into the latch and then a DATA READY signal (DATA_R) is generated and sent to the computer.
- Again the computer waiting for the DATA_R signal proceeds further and activates the OE signal and then reads the data from the latch.
- After this the computer program deactivates the nOE signal first and the EXT0 signal later.
- The STATUS flag is cleared at the start of the external interrupt subroutine and the DATA_R flag is cleared at the start of the serial interrupt subroutine.
- The RE_DE flags remains low till the end of the program as the microcontroller has to keep on storing the incoming serial data and is not expected to transmit any data.

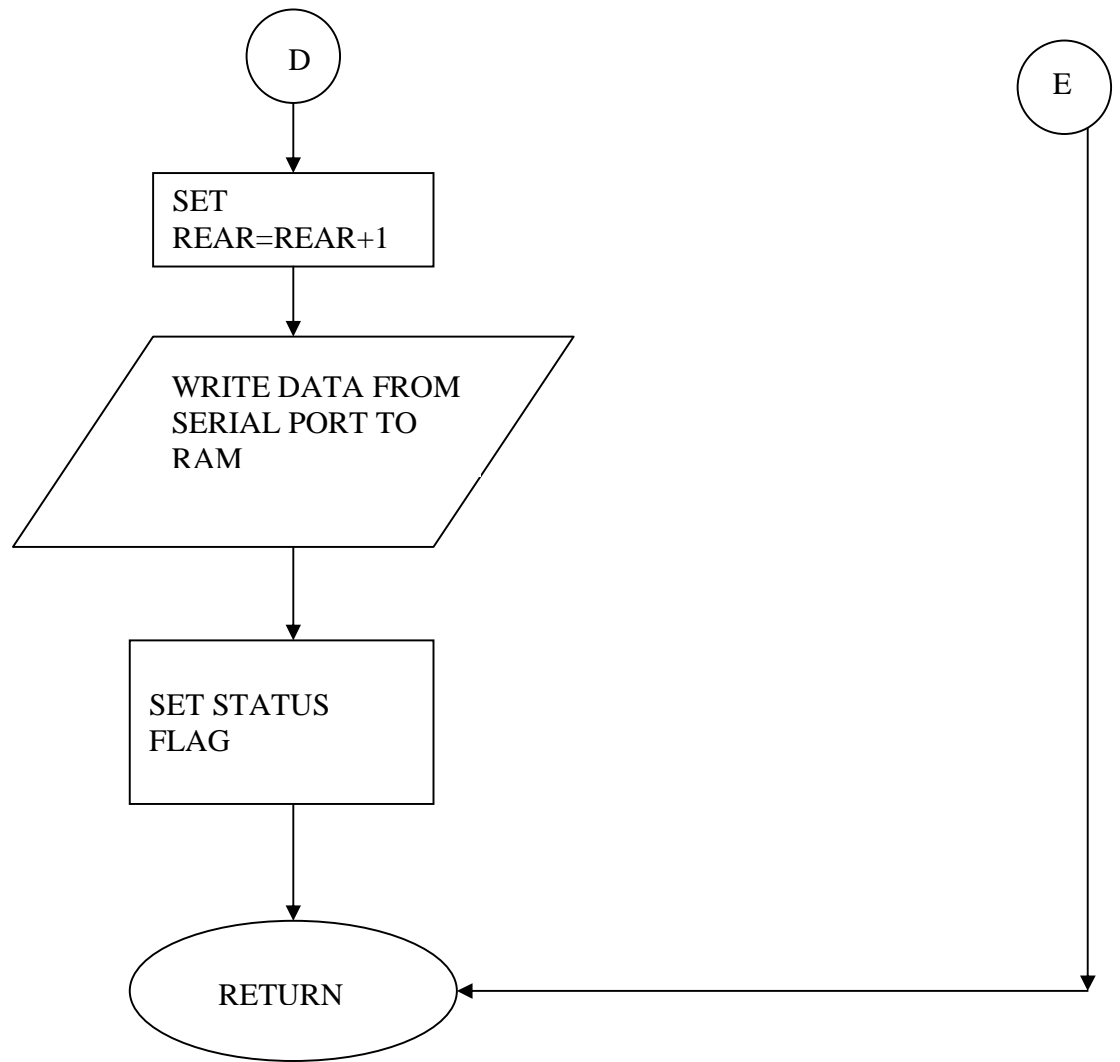
C. FLOWCHART

Here is a flowchart that explains the execution of the program.









D. THE CODE

```

DPL1 EQU 84H
DPH1 EQU 85H
DPS EQU 86H
SCON0 EQU 98H
SBUF0 EQU 99H
ACCU EQU 0E0H
DE_RE BIT p1.7 ; ACTIVE
LOW RECIEVE ENABLE AND ACTIVE
HIGH DATA ENABLE
STATUS BIT p1.6
DATA_R BIT p1.5
NULL EQU 5000H
START EQU 5001H
END EQU 7FFFH
  
```

```

org 4003h
ljmp isr_exti
  
```

```

org 4023h
ljmp isr_seri
  
```

```

org 4200h
;dptr=rear,dptr1=front
main: CLR P1.6
      mov DPS,#0
      mov dptr,#NULL ;LOAD
      NULL INITIALLY IN FRONT NAD REAR
      mov DPS,#1
      mov dptr,#NULL
      mov DPS,#0
      mov tmod,#20h ;TIMER 1
      MOBE 2 AUTO RELOAD
  
```

```

        mov th1,#0f3h ; LOAD -13
FOR 4800 BAUD RATE
        mov scon,#50h ; 8-BIT , 1
START , 1 STOP BIT
        mov a,pcon ;USING
PCON TO GET DOUBLE RATE
        setb acc.7 ;OF 9600
BAUD/SEC WE ARE STILL WORKING IN
TESTING MODE AND HENCE USING
9600baud/sec INSTEAD OF 750kbaud/sec
        mov pcon,a
        mov ie,#10010001b ;ENABLE
SERIAL AND EX0 INTERRUPT
        mov ip,#00010000b ;ASSIGN
PRIORITY TO SERIAL INTERRUPT
        setb tcon.0 ;EDGE
TRIGGERED EXTERNAL INTERRUPT
        clr STATUS ;CLEAR
PRINTER PORT STATUS FLAG
        clr DE_RE ;ENABLE
RECIEVE OF 75176
        clr DATA_R ;DISABLE
DATA READY
        setb tr1
now:    sjmp now

isr_seri: clr DATA_R ;CLEAR
FLAG,DATA IS NOW BEING WRITTEN
        push DPS ;SAVING
DPS AND A
        push ACCU
        jb ti,trans ;CHECK FOR
TRANSMIT INTERRUPT
        mov a,sbuf
        mov r1,a ;SAVE
INCOMING SERIAL DATA IN R1
        mov r6,dph ;CHECK IF
REAR=END AND
        cjne r6,#7fh,next2
;FRONT=START
        mov r6,dpl ;IF SO THEN
THERE IS OVERFLOW
        cjne r6,#0ffh,next2
        mov r6,DPH1
        cjne r6,#50h,next2
        mov r6,DPL1
        cjne r6,#01h,next2
        mov a,#0 ;RETURN 0
TO INDICATE OVERFLOW
here1:  sjmp here1
next2:  mov a,dph ;CHECK IF
FRONT=REAR+1
        mov r6,DPH1 ;IF SO
THEN AGAIN THERE

```

```

        cjne a,6,next3 ;IS
OVERFLOW
        mov a,dpl
        inc a
        mov r6,DPL1
        cjne a,6,next3
        mov a,#0 ;RETURN 0
TO INDICATE OVERFLOW
here2:  sjmp here2
next3:  mov r6,DPH1 ;CHECK IF
FRONT=NULL
        cjne r6,#50h,nt ;IF SO THEN
LOAD FRONT=START
        mov r6,DPL1 ;AND
REAR=START
        cjne r6,#00h,nt
        inc dptr
        mov DPS,#1
        inc dptr
        mov DPS,#0
nt:     mov r6,dph ;CHECK IF
REAR=END
        cjne r6,#7fh,next ;IF SO
THEN LOAD REAR=START
        mov r6,dpl
        cjne r6,#0ffh,next
        mov dptr,#START
        sjmp for
next:   inc dptr ;ELSE REAR=REAR+1
for:   mov a,r1 ;LOAD SAVED VALUE
FROM SERIAL PORT
        movx @dptr,a ;LOAD
VALUE INTO RAM
        setb STATUS ;SET
STATUS FLAG, DATA IS AVAILABLE
        clr ri ;CLEAR RI FLAG
bac:   pop ACCU ;RELOAD VALUES
OF A AND DPS
        pop DPS
        reti
trans:  clr ti ;CLEAR TI FLAG ON
TRANSMIT
        sjmp bac
;INTERRUPT AND RETURN BACK

        org 4300h
isr_exti:
        push DPS ;SAVE VALUES
OF A AND DPS
        push ACCU
;FRONT=DPTR1 , REAR=DPTR1
        mov r7,DPH1 ;CHECK IF
FRONT=NULL

```

```

                cjne r7,#50h,nt1 ;IF SO
THEN THERE IS UNDERFLOW
                mov r7,DPL1
                cjne r7,#00h,nt1
                sjmp nxt1 ;HENCE
RETURN BACK
nt1:   mov DPS,#1
                movx a,@dptr ;LOAD
VALUE FROM RAM
                push DPH1 ;AND SEND
IT TO LATCH
                push DPL1
                mov dptr,#8000h
                movx @dptr,a
                setb DATA_R ;SET
DATA READY FLAG
                pop DPL1 ;AS DATA IS
READY TO BE READ FROM LATCH
                pop DPH1 ;CHECK IF
FRONT=REAR
                mov a,dph ;THEN LOAD
FRONT=NULL
                mov r7,DPH1 ;AND
REAR=NULL
                cjne a,7,nx

```

```

                mov a,dpl
                mov r7,DPL1
                cjne a,7,nx
                mov dptr,#NULL
                mov DPS,#0
                mov dptr,#NULL
                mov DPS,#1
                sjmp nxt1
nx:   mov r7,DPH1 ;CHECK IF
FRONT=END
                cjne r7,#7fh,nxt ;LOAD
FRONT=START
                mov r7,DPL1
                cjne r7,#0ffh,nxt
                mov dptr,#START
                sjmp nxt1
nxt:  inc dptr ;ELSE
FRONT=FRONT+1
nxt1: pop ACCU ;RELOAD A AND
DPS
                pop DPS
                clr STATUS ;DATA
HAS BEEN READ CLEAR STATUS FLAG
                reti

```

IX FINAL INTERFACING PROGRAM USING VISUAL C++

A. EXPLANATION

The final program to interface the circuit has been made in visual C++. It makes use of the `_inp()` and `_outp()` functions to access the various registers of the parallel port which is available in the header file `stdlib.h`. The basic

B.)THE CODE

```

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <string.h>
/*#include "chipio.c"*/ /*THIS FILE
SUPPORTS inportb() AND outportb() USED
IN TURBOC*/

#define PORTADDRESS 0x378
/*ADDRESSES OF VARIOUS REGISTERS
OF PARALLEL PORT*/
#define DATA PORTADDRESS+0
//DATA REGISTER

```

working of the program is that it store the incoming parallel port data in various batch files each of a limit of 10,000 bytes and such 1000 files can be created by the program. Also the capacity of each file and the number of files can be changed according to requirements. The name given to the various text files is of the form data0.txt, data1.txt and so on till data999.txt. The remaining algorithm of the program is similar to the one made in turbo C.

```

#define STATUS PORTADDRESS+1;
//STATUS REGISTER
#define CONTROL PORTADDRESS+2;
//CONTROL REGISTER

void main(void){
    long int i=0,k=0; //i IS FOR LOOP
    COUNTER AND k IS FILE NUMBER
    int exitloop=0;
    char
    j[3]="0",filename[20]="data",tempfilename[20]=
    "data";
    FILE *fp;
    _outp(CONTROL, _inp(CONTROL)&0
    xDF);

```

```

        _outp(CONTROL,_inp(CONTROL)|0x
2C);/*ACTIVATE BIDIRECTIONAL DATA
TRANSFER*/
        while(1)
        {
                strcat(tempfilename,j);
//GENERATE NEW FILENAME
                fp=fopen(tempfilename,"a+");
//OPEN FILE
                for(i=0;i<10000;i++){

                        if((_inp(STATUS)&0x08)!=0){
//CHECK STATUS FLAG
                                _outp(CONTROL,_inp(CONTROL)&0
xF7);/*GENERATE EXTERNAL
INTERRUPT*/
                                if((_inp(STATUS)&0x10)==0){ //CHECK
DATA READY FLAG
                                        _outp(CONTROL,_inp(CONTROL)&0
xFB);//ENABLE LATCH OUTPUT
                                        fprintf(fp,"DATA:
%x\n",_inp(DATA)); //TRANSFER DATA
TO FILE
                                        _outp(CONTROL,_inp(CONTROL)|0x
04); //DISABLE LATCH OUTPUT

```

X REFERENCES

- [1] Parallel Port Complete By Jan Axelson
- [2] 8051 microcontroller and embedded systems by Muhammad Ali Mazidi and Janice Gillespie Mazidi
- [3] <http://www.arcelect.com/rs232.htm>
- [4] <http://www.beyondlogic.org/spp/parallel.htm>

```

        _outp(CONTROL,_inp(CONTROL)|0x
08); //SET INTERRUPT PIN
                }
        }
        fclose(fp); //CLOSE
FILE
        ++k;
//INCREMENT FILE NUMBER
        itoa(k,j,10); //CONVERT
INTEGER FILE NUMBER TO STRING
        strcpy(tempfilename,filename);
        if(kbhit() //EXIT WHEN
ESCAPE IS HIT ON KEYBOARD
        {
                exitloop=getch();
                if(exitloop==27)
                        break;
        }
        }
        getch();
}

```

- [5] <http://www.lammertbies.nl/comm/info/RS-485.html>