Technical Note

Using eZ80Acclaim!® Real Time Clock

TN001603-0607

General Overview

ZiLOG's eZ80Acclaim!® devices contain an on-chip real time clock (RTC) that is useful in applications that keep track of time, even during power failure. An input pin, RTC_V_{DD}, isolated from other power pins on the eZ80Acclaim! device, provides power to the RTC block. This pin can be driven by battery power in order to maintain RTC functionality during power failure.

This Technical Note provides a brief description of the RTC function. The source code associated with this document, TN0016-SC01.zip, is available on www.zilog.com.

Table 1 lists 14 RTC registers available to configure the RTC peripheral. The RTC registers are programmed for both real time and alarm conditions.

Table 1. Real Time Clock Registers

RTC Register	Register Description	Hexadecimal Address
RTC_SEC	RTC Seconds register	E0
RTC_MIN	RTC Minutes register	E1
RTC_HRS	RTC Hours register	E2
RTC_DOW	RTC Day-of-the-week register	E3
RTC_DOM	RTC Day-of-the-month register	E4
RTC_MON	RTC Month register	E5
RTC_YR	RTC Year register	E6
RTC_CEN	RTC Century register	E7
RTC_ASEC	RTC Alarm Seconds register	E8
RTC_AMIN	RTC Alarm Minutes register	E9
RTC_AHRS	RTC Alarm Hours register	EA
RTC_ADOW	RTC Alarm Day-of-the-week register	EB
RTC_ACTRL	RTC Alarm Control register	EC
RTC_CTRL	RTC Control register	ED

When executing the sample code associated with this document, the software receives its input from one of the eZ80Acclaim! device serial ports. As a result, you are allowed to program the RTC registers from any terminal-based application (for example, HyperTerminal application).

Set the HyperTerminal connectivity settings to the following values:

Bits per second	57600
Data bits	8
Parity	None
Stop Bits	1
Flow Control	None

After a power-on reset, the following sign-on message is displayed in the HyperTerminal window:

```
Reset

?  Display this help menu
s  Set real time clock
r  Read RTC value

Acclaim>
```

To set RTC values, type `s` at the `Acclaim` prompt. The day, month, year, and the current time in hours and minutes are required as inputs at each prompt that follows. These inputs are then loaded into each register of the RTC block. Type `r` to read RTC value from the RTC registers.

Source Code

This section lists the source code to set and read the eZ80Acclaim!® RTC.

```
// Define Months and Days

#define JAN 1
#define FEB 2
#define MAR 3
#define APR 4
#define MAY 5
#define JUN 6
#define JUL 7
#define AUG 8
#define SEP 9
#define OCT 0x10
#define NOV 0x11
#define DEC 0x12

#define MO  1
#define TU  2
#define WE  3
#define TH  4
```

```
    #define FR  5
    #define SA  6
    #define SU  7

int rtc_buff[20];               // RTC ascii buff

//*******************************************************
// Make sure you run your hardware/software initialization routines
// before entering main
// Note: The Get() and Put() serial port routines are not included
// in this list. Refer ZiLOG's eZ80 Flash Loader source code for
// other routines.
//
void main(void)
{
int c;
int am_flag;
int baudrate;

RTC_CTRL = 0x20;                // Make sure real time clock is off

am_flag=0;                      // Reset real time clock am/pm flag

baudrate=0x36;                  // 57600 50M/(16*57600)= 54 = 36H
init_com0(baudrate);            // Init com 0

delay();                        // Just a little delay
_ei();                          // Turn on all interrupts

puts("\r\nReset\r\n");          // Output to com0 ascii message
             delay();           // Delay a little
display_help();

prompt();


//*******************************************************
//Get command from com0 and run routines
//
  do {
    do {
      c = getchar();
    } while(c < 0);

    if(c == '\n' || c == ' ')
      continue;

    putchar(c);                 // Get a Char. For the serial port
                                // Refer ZiLOG's Flash Loader source code for
                                // this routine
    process_char(tolower(c));
```

```
  } while(1);

}// end of main

/*************************************************************/
//    This parses the commands received from main() function, sorts
//    through, determine what type of command it is, then call
//    the appropriate routine to handle it.

void process_char(int c)
{

  switch( tolower(c) )
  {
    case '?':
        display_help();
        break;

    case 's':
     RTC_CTRL = 0x21;           // unlock RTC - you must do this before
                                // you can write to the RTC registers
    get_user_input();           // Get users input from Serial port

    RTC_DOW = rtc_buff[0];      // Move data to RTC registers
    RTC_DOM = rtc_buff[1];
    RTC_MON = rtc_buff[2];
    RTC_YR  = rtc_buff[3];
    RTC_HRS = rtc_buff[4];
    RTC_MIN = rtc_buff[5];
    RTC_SEC = rtc_buff[6];

    RTC_CTRL = 0x20;            // lock RTC to start clock running again

    break;

    case 'r':
      Display_RTC();            // Output the RTC values on the console
    break;

    default:
    puts("\r\nUnknown Command");
    break;

  }

  prompt();

} // end process_char


/*************************************************************
```

```
 * The display_help() function displays the list of valid commands.
 *************************************************************/
void display_help(void)
{
  puts("\r\n");
  puts("\r\n  ?\tDisplay this help menu");
  puts("\r\n  s\tSet real time clock");
  puts("\r\n  r\tRead RTC value\r\n");
  return ;

}/** end of display_help() */
/*************************************************************
 * This displays the system prompt
 *************************************************************/
void prompt(void)
{

  puts("\r\nAcclaim>");

  return ;

}/** end of prompt() */


/*************************************************************
 * Get user input for RTC
 * Note: No error checking on input, if error just re-enter
 *************************************************************/
void get_user_input(void)
{
  char c;

  puts("\r\n");
  puts("\r\nSet Real time Clock");
  puts("\r\n");
  puts( "\r\nEnter Day of the Week (Monday = 1, Sunday = 7)" ) ;

  puts("\r\nDay of Week: ");

  c=getchar();
  putchar(c);

// Save Day of the Week
  rtc_buff[0] = c -0x30;

  puts( "\r\nEnter Month (Jan = 01, Dec = 12)" ) ;

  puts("\r\nMonth: ");

  c=getchar();
  putchar(c);
```

```
  rtc_buff[2] = ((c-0x30) << 4);
  c=getchar();
  putchar(c);

//Save Month
  rtc_buff[2] = rtc_buff[2] | (c-0x30);

  puts( "\r\nEnter Day (01-31)" ) ;

  puts("\r\nDay: ");

  c=getchar();
  putchar(c);
            rtc_buff[1] = ((c-0x30) << 4);
  c=getchar();
  putchar(c);

//Save Day
  rtc_buff[1] = rtc_buff[1] | (c-0x30);

  puts( "\r\nEnter Year (01-99)" ) ;

  puts("\r\nYear: ");

  c=getchar();
  putchar(c);
  rtc_buff[3] = ((c-0x30) << 4);
  c=getchar();
  putchar(c);

//Save Year
  rtc_buff[3] = rtc_buff[3] | (c-0x30);

  puts( "\r\nHours (01-24)" ) ;

  puts("\r\nHours: ");

  c=getchar();
  putchar(c);
  rtc_buff[4] = ((c-0x30) << 4);
  c=getchar();
  putchar(c);

//Save Current Hour
  rtc_buff[4] = rtc_buff[4] | (c-0x30);

  puts( "\r\nMins (00-59)" ) ;

  puts("\r\nMins: ");

  c=getchar();
```

```
  putchar(c);
  rtc_buff[5] = ((c-0x30) << 4);
  c=getchar();
  putchar(c);

//Save Current Mins.
  rtc_buff[5] = rtc_buff[5] | (c-0x30);

//Just Reset Seconds.
  rtc_buff[6] = 0;

  puts("\r\n");
  return ;
}

/*************************************************************
 * This displays Real time Clock on console
 *************************************************************/
void Display_RTC(void)
{
  int c;
  char buffx[3];

  c = RTC_DOW;

  switch (c)
  {
  case MO:
    puts ("MON");
    break;

  case TU:
    puts ("TUE");
    break;

  case WE:
    puts ("WED");
    break;

  case TH:
    puts ("THU");
    break;

  case FR:
    puts ("FRI");
    break;

case SA:
    puts ("SAT");
    break;
```

```
  case SU:
    puts ("SUN");
    break;

  default:
    break;

puts(" ");

c = RTC_MON;
  switch (c)
  {
  case JAN:
    puts("JAN");
    break;

  case FEB:
    puts ("FEB");
    break;

  case MAR:
    puts ("MAR");
    break;

  case APR:
    puts ("APR");
    break;

  case MAY:
    puts ("MAY");
    break;

  case JUN:
    puts ("JUN");
    break;

  case JUL:
    puts ("JUL");
    break;

  case AUG:
    puts ("AUG");
    break;

  case SEP:
    puts ("SEP");
    break;

  case OCT:
    puts ("OCT");
    break;
```

```
case NOV:
  puts ("NOV");
  break;

case DEC:
  puts ("DEC");
  break;

  default:
  break;
 }

puts("-");

*buffx='\0';
c = RTC_DOM;
bin_to_ascii(c, buffx);
puts(buffx);
puts("-");
*buffx='\0';
c = RTC_YR;
bin_to_ascii(c, buffx);
puts(buffx);
puts(" ");
*buffx='\0';
c = RTC_HRS;
if (c > 0x12)
  {
  am_flag=0;
  if (c == 0x20)
    {
     c= 0x08;
    }
  else if (c == 0x21)
    {
    c= 0x09;
    }
  else
    {
    c=c-0x12;
    }
  }
 else
  {
  am_flag=1;
  }
bin_to_ascii(c, buffx);
puts(buffx);
puts(":");
*buffx='\0';
```

```
  c = RTC_MIN;
  bin_to_ascii(c, buffx);
  puts(buffx);
  puts(":");
  *buffx='\0';
  c = RTC_SEC;
  bin_to_ascii(c, buffx);
  puts(buffx);
}
/*************************************************************/
// This will convert the binary value c into its ascii
// representation in hexadecimal. It will append the two ascii characters at
// the end of *buff
// If you want to save it at the start of buff, make
// buff[0]='\0'; This function will also null terminate
// the string.

void bin_to_ascii(unsigned char c, char *buff)
{
  while(*buff)
            {
    buff++;
  }

  if( ((c >> 4) & 0x0f) <= 9)
           {
    *buff++ = ((c >> 4) & 0x0f) + '0';
  } else {
    *buff++ = ((c >> 4) & 0x0f) + 'A' - 10;
  }

  if( (c & 0x0f) <= 9)
            {
    *buff++ = (c & 0x0f) + '0';
  } else {
    *buff++ = (c & 0x0f) + 'A' - 10;
  }
  *buff='\0';
}
/*************************************************************/
```

Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

ZiLOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZiLOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer