HACKADAY.IO

# Added Real Time Clock

A project log for 3-Chip Z80 Design
*Combining a Z80 retro design with a modern PSoC CPU.*

land-boards.com • 10/26/2019 at 11:27 • 0 Comments

---

Added support for the Real Time Clock (RTC). The Z80_PSOC board has a spot for an optional 32.768 KHz watch crystal which provides accurate time.

The PSoC has  internal circuitry for a clock. The board also has a battery holder so that the RTC is maintained when the main power is removed from the board.

### RTC Access from the Z80

I added access from the Z80 to the Real Time Clock functions. The RTC addresses can be located at any 2 locations in the I/O Space. I put these addresses into the file HardwareConfig.h as:

```
#define RTC_DATA    0x60    // 96 dec
#define RTC_CSR     0x61    // 97 dec
```

The first location, RTC_DATA is the RTC Data location. The other location, RTC_CSR is the control/status register for the RTC.

The code for the clock is in Z80_RTC.c and .h files. The files implement a simple state machine to set and track the state of the clock interface. The states are in an enumeration as:

```
enum rtcStates
{
    RTC_SEC,    // 0
    RTC_MIN,    // 1
    RTC_HR,     // 2
    RTC_DAY,    // 3
    RTC_MON,    // 4
    RTC_YR_HI,  // 5
    RTC_YR_LO   // 6
};
```

The initialization code sets the first access to the seconds and turns on the clock. After reading the RTC_DATA from the code, the second access will automatically be set to the minutes. This continues on through the seven values. After the last location is read/written the state machine will go back to seconds.

Writing to the RTC_CSR location sets the offset to any of the fields. Values need to be limited to 0-6 matching the field offset. Reading the RTC_CSR location returns the the current state pointer.

### Setting the RTC from BASIC

Setting the RTC code looks like this:

```
5 REM SET THE CLOCK POINTER TO SECONDS
10 OUT 97,0
20 REM SEC
30 OUT 96,1
40 REM MIN
50 OUT 96,35
60 REM HOUR
70 OUT 96,8
80 REM DAY
90 OUT 96,25
100 REM MON
110 OUT 96,10
120 REM YR_LO
130 OUT 96,227
140 REM YR_HI
150 OUT 96,7
```

BASIC uses integer math so the values are simple integers. The only complicated part is setting the year since it's a 16-bit value and has to be set in 2 parts. Setting the upper byte of the year to 7 sets the clock to 7*256=1792. Adding 227+1792 sets the year to 2019.

### Reading the RTC from BASIC

The code to read the RTC is similar:

```
400 OUT 97,0
410 SC = INP(96)
420 MN = INP(96)
430 HR = INP(96)
440 DY = INP(96)
450 MO = INP(96)
460 YL = INP(96)
470 YH = INP(96)
480 PRINT "YR";((YH*256)+YL);"MON";MO;"DAY";DY;"TIME";HR;MN;SC
```

When RUN 400 is entered and looping, the clock returns:

```
YR 2019 MON 10 DAY 27 TIME 20  16  17
YR 2019 MON 10 DAY 27 TIME 20  16  17
YR 2019 MON 10 DAY 27 TIME 20  16  17
YR 2019 MON 10 DAY 27 TIME 20  16  18
```

The hours is a 24 hour (Military style) clock. Subtracting 12 from numbers over 12 and adding an AM/PM indication could be done.

### PSoC Code Implemetation

As mentioned there is a driver Z80_RTC.c and .h that maps the Z80 accesses to the RTC code generated from the PSoC API generator. Setting the value works like this:

```
////////////////////////////////////////////////////////////////////////////
// void writeRTC(uint8) - Write to RTC
// Auto-increment to the next field

void writeRTC(void)
{
    uint16 year;
    uint16 year2;
    uint8 wrVal = Z80_Data_Out_Read();

    switch (rtcState)
    {
        case RTC_SEC:
            RTC_WriteSecond(wrVal);
            rtcState = RTC_MIN;
            break;
        case RTC_MIN:
            RTC_WriteMinute(wrVal);
            rtcState = RTC_HR;
            break;
        case RTC_HR:
            RTC_WriteHour(wrVal);
            rtcState = RTC_DAY;
            break;
        case RTC_DAY:
            RTC_WriteDayOfMonth(wrVal);
            rtcState = RTC_MON;
            break;
        case RTC_MON:
            RTC_WriteMonth(wrVal);
            rtcState = RTC_YR_LO;
            break;
        case RTC_YR_LO:
            year = wrVal;
            RTC_WriteYear(year);
            rtcState = RTC_YR_HI;
            break;
        case RTC_YR_HI:
            year = RTC_ReadYear() + (wrVal<<8);
```

```
            RTC_WriteYear(year);
            rtcState = RTC_SEC;
            break;
    }
    ackIO();
}
```

Reading the RTC is driven by the same state bits and the code looks like this:

```
////////////////////////////////////////////////////////////////////////////
// uint8 readRTC() - Read RTC
// Auto-increment to the next field

void readRTC(void)
{
    uint8 retVal = 0;
    switch (rtcState)
    {
        case RTC_SEC:
            retVal = RTC_ReadSecond();
            rtcState = RTC_MIN;
            break;
        case RTC_MIN:
            retVal = RTC_ReadMinute();
            rtcState = RTC_HR;
            break;
        case RTC_HR:
            retVal = RTC_ReadHour();
            rtcState = RTC_DAY;
            break;
        case RTC_DAY:
            retVal = RTC_ReadDayOfMonth();
            rtcState = RTC_MON;
            break;
        case RTC_MON:
            retVal = RTC_ReadMonth();
            rtcState = RTC_YR_LO;
            break;
        case RTC_YR_LO:
            retVal = (uint8)(RTC_ReadYear() & 0xff);
            rtcState = RTC_YR_HI;
            break;
        case RTC_YR_HI:
            retVal = (uint8)(RTC_ReadYear() >> 8);
            rtcState = RTC_SEC;
            break;
    }
    Z80_Data_In_Write(retVal);
    ackIO();
}
```

The source code is in GitHub.

Previous Log
Z80 (CP/M) Writing to SD Card
10/25/2019 at 11:06 • 0 comments

Next Log
More SD Card Write Details
10/26/2019 at 14:10 • 0 comments

## DISCUSSIONS

*Log In or become a member to leave your comment*

Log In/Sign up to comment

Going up?

About Us Contact Hackaday.io Give Feedback Terms of Use Privacy Policy Hackaday API