Entry CK78764

Wireless TIme Server (Abstract)

The Wireless Time Server is a solar powered device which receives time and date data from a GPS receiver module and rebroadcasts it into a mesh network through an XBee module for use by any devices attached to the network.

The server is powered by a small NiCd battery pack which is recharged by a small solar photovoltaic panel. The circuitry and software in the server have been designed to ensure that the absolute minimum amount of power is used, and that the server will remain operational over night until charging can resume at sunrise.

The server transmits its data to the mesh network over an XBee ZNet module which is configured as an End Node to the network. The operating mode of the server is selected by sending control characters back in over the same wireless network connection.

The server has three operating modes. In NORMAL mode, the server broadcasts the time and date once per minute. In normal mode the server is in low-power sleep for 55 seconds out of each minute. If another device on the network requests it (i.e. an antenna positioner), the server can switch into STREAM mode where the time and date are transmitted once per second and the server never goes to sleep. If placed in the TEST mode, the voltage of the battery pack is measured and transmitted along with the time and date data once per second.

The project is built into a commonly available and inexpensive weather tight PVC electrical junction box. Since I've installed the server atop a 20-foot tower, I have painted it bright white both to protect it from solar heating, and UV exposure which can rapidly degrade the PVC when it is exposed to sunlight.

While the software used in the server is relatively short and simple, it serves as a good demonstration of programming in a high level language (MPIDE) with a focus on saving time and conserving energy.

As presented the Wireless Time Server achives my design goals, however custom functionality could easily be added by another builder. For example many other measurements could be added to the TEST mode (charging current, temperature, signal strength to name a few).

CODE SAMPLE

```
/*
  WIRELESS TIME SERVER
  This code receives a data sentence from a GPS module, parses out the time and date information,
  reassembles the information into a new string, and transmits it through an XBee module.

 */
char SysMode;      // N : Normal Operation (1 transmission per minute)
                   // S : Stream Mode (continuous transmission)
                   // T : Test Mode (continuous data transmission with measurements)
int  pointer = 0;  // General use pointer
int  ptr = 0;      // General use pointer
```

```
int  LoopCnt = 0;  // General use loop counter

// Variables and constants for battery voltage measurement

int  VBatt     = 5; // Enable VBatt test pin
int  ResetTrig = 6; // Trigger reset by setting HIGH
int  BatVoltage= 0; // Battery voltage reading

// Variables and constants for the XBee module

int  XBeeSleep = 2; // XBee module sleep pin
int  XBeeReset = 3; // XBee module reset pin

// Variables and constants for the GPS module

int  GPSpower  = 4; // Power control to GPS Module
int  byteCnt = 0;   // General use counter
int  inByte = 0;    // Character read from GPS
char inGPS[70];     // GPS sentence
char inTime[6];     // Time from GPS as HHMMSS
char inDate[6];     // Date from GPS as DDMMYY
char inStat;        // GPS status A=valid V=not valid

void setup() {
  SysMode = 'N';  // Set to default operation mode - NORMAL

// Set up battery voltage measurement

  pinMode(VBatt, OUTPUT);       //  HIGH=on  LOW=off
  digitalWrite(VBatt, LOW);     //  Turn off VBatt measurement

// Set up GPS module

  pinMode(GPSpower, OUTPUT);    //  HIGH=on  LOW=off
  digitalWrite(GPSpower, HIGH); //  Turn on GPS module
  delay(1000);                  //  Give GPS a second to wake up

  /*
  The GPS module can communicate at 4800, 9600, 19200, or 38400 baud. We want the module
  set to the highest speed, but can not absolutely predict what the current baud rate is.
  We transmit the configuration string to set the baud rate to 38400 at each of the three
  lower baud rates. If the module is already at 38400 when we start, it will just ignore
  the configuration strings.
  */
  Serial2.begin(4800);
  Serial2.write("$PSRF100,1,38400,8,1,0*3D\r\n");
  Serial2.end();
  delay(500);
```

```
 Serial2.begin(9600);
 Serial2.write("$PSRF100,1,38400,8,1,0*3D\r\n");
 Serial2.end();
 delay(500);
 Serial2.begin(19200);
 Serial2.write("$PSRF100,1,38400,8,1,0*3D\r\n");
 Serial2.end();
 delay(500);
 Serial2.begin(38400);

 /*
 By default, the GPS module transmits 3 sentences once per second, and another sentence twice per
 second. Since the only need one of the sentences the following three lines disable the other
 three (saving time and power). Then the module will only transmit one sentence (GPRMC) once
 per second.
 */
 Serial2.write("$PSRF103,0,0,0,1*24\r\n");     // Disables GPGGA sentence
 Serial2.write("$PSRF103,2,0,0,1*26\r\n");     // Disables GPGSA sentence
 Serial2.write("$PSRF103,3,0,0,1*27\r\n");     // Disables GPGSV sentence

 //  SET UP XBEE MODULE ----------------------------------------------------------------

 Serial3.begin(9600);         //  xBee port
 pinMode(XBeeSleep, OUTPUT);   //  HIGH=Asleep  LOW=Awake
 pinMode(XBeeReset, OUTPUT);   //  Pull LOW to reset XBee
 digitalWrite(XBeeSleep, LOW); //  Ensure XBee is awake
 /* On very rare ocasions I had the XBee fail to initialize on power up. Adding
 this quick reset seemed to solve the issue. */
 digitalWrite(XBeeReset, LOW); //  Assert XBee Reset
 delay(10);                    //      for 10 ms
 digitalWrite(XBeeReset, HIGH);//  Release XBee from reset

}
```

PHOTOS

PHOTO 1

PHOTO 2

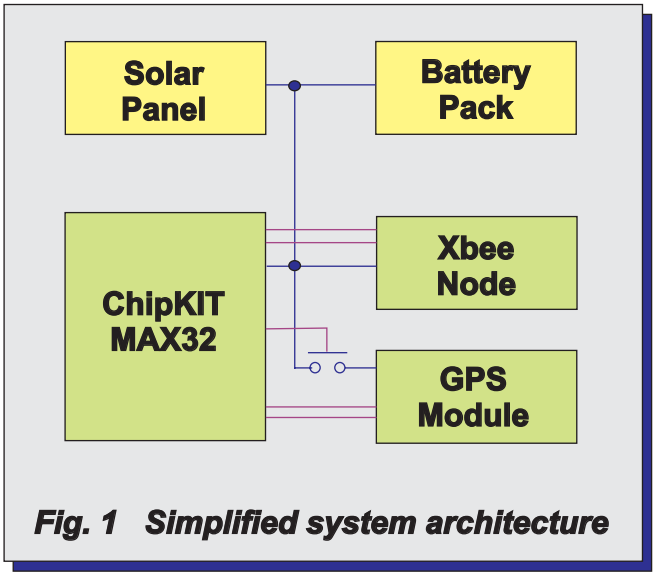
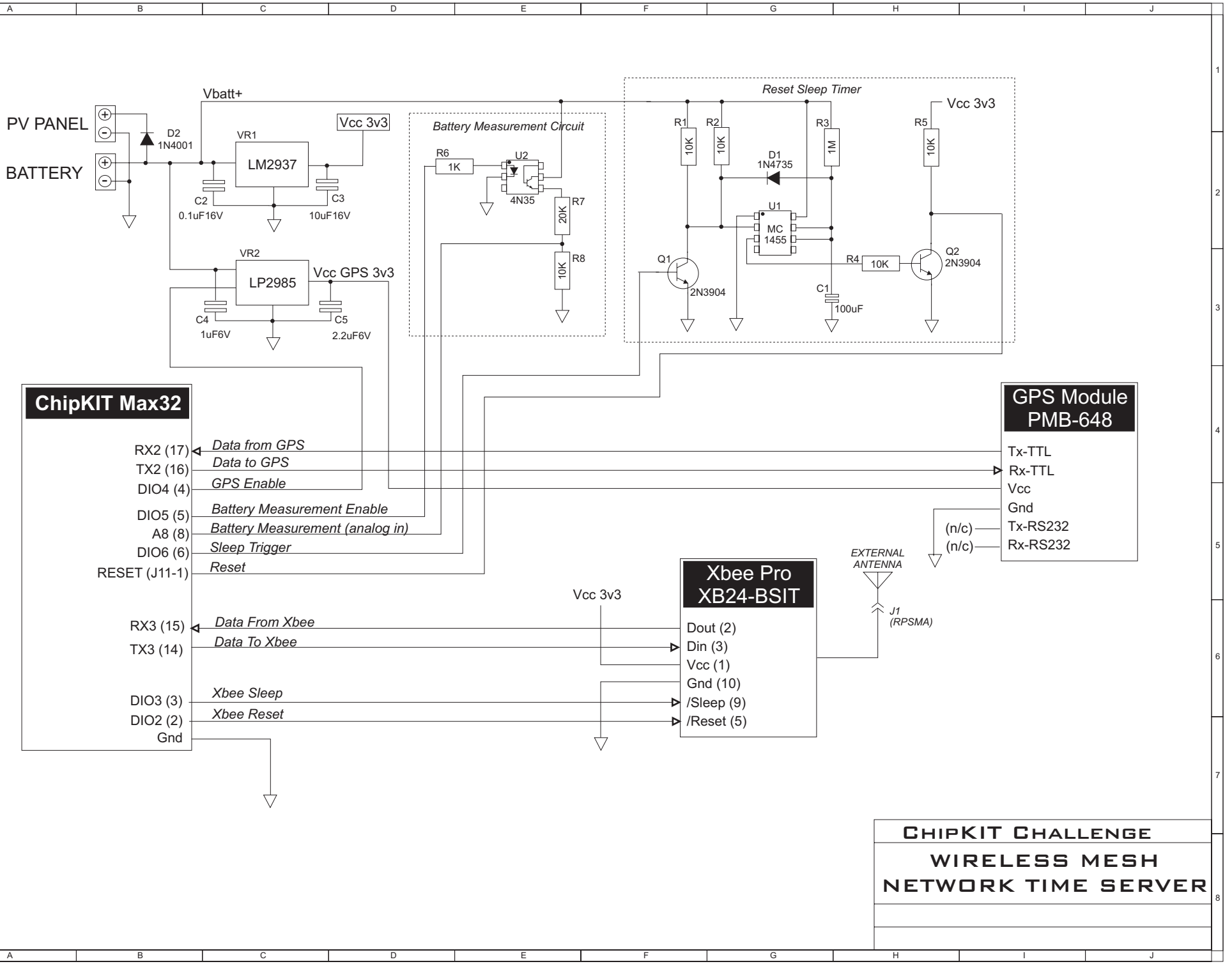


Fig. 1 Simplified system architecture



CHIPKIT CHALLENGE

WIRELESS MESH NETWORK TIME SERVER