DesignSpark chipKIT™ Challenge

Entry # 73465

Eco Friendly 24/7 automation and monitoring harvesting controller.

A device that is able to monitor and control a house should be always present if someone is looking to conserve some energy. In a typical household we will also found a security system. Most of the time you will not find both system mixed in a single compact device for a lower price. That is the aim of this project.
A common home automation device will have some modern type of communication in order to allow user access whatever the user location is. Similar it will have some type of connection to local and remote sensors that will allow it to monitor and control appliances. A security system do more or less the same but with focus on theft intrusion detection and 24/7 operation.

The solution.

The system is built around the chipKit MAX32 arduino compatible board which contains a PIC32 device. The uninterrupted operation is achieve by a custom designed chipSolar board which provides power to the Max32 board from two Li-Ion cells of 17.3Wh of capacity, this board have the same footprint and connects on the top of the Max32, while passing signals to allow additional chipKit daughter boards to plug in. The Eco-friendly operation starts here as the chipSolar board allow to charge the battery from a solar panel. The board implements an MPPT charger which allow to deal with the non linear output efficiency of a solar panel. The decision was to build a robust system, in that sense the MPPT charger design uses analog circuitry allowing to spend CPU cycles in other critical tasks. Only signals that indicate charge and fault conditions are handle by the arduino. A 10W solar panel capable of provide 8V at 1A was tested, charging the 2 Li-Ion cells in less than 3 hours in a shinny day.

To provide the connectivity needed a custom chipWireless board was designed. It have a Quad Band GSM/GPRS modem, an Xbee socket, SD card connector and low cost hardware companion RTCC.

The software was written using MPIDE. The system starts doing a memory verification and initialization, with the help of the non volatile SRAM memory provided by the RTCC companion IC using a custom library.
After verifying the previous operation state it continues with the modem initialization in order to be ready for SMS command reception and be able to send data to a server. The server used was Pachube, which allow to connect people to devices in a friendly and easy way making further analysis of the data and integration with other applications very easy.
A ZigBee network is established with the help of an XBee device pre-programmed as a coordinator. The software check for devices that have joined the network and add them to a joined list. By implementing a simple sensor library for XBee End Devices it's possible to add nodes to the network without re-programming the firmware.
An SD card is used to log data from sensors and future memory requirements.

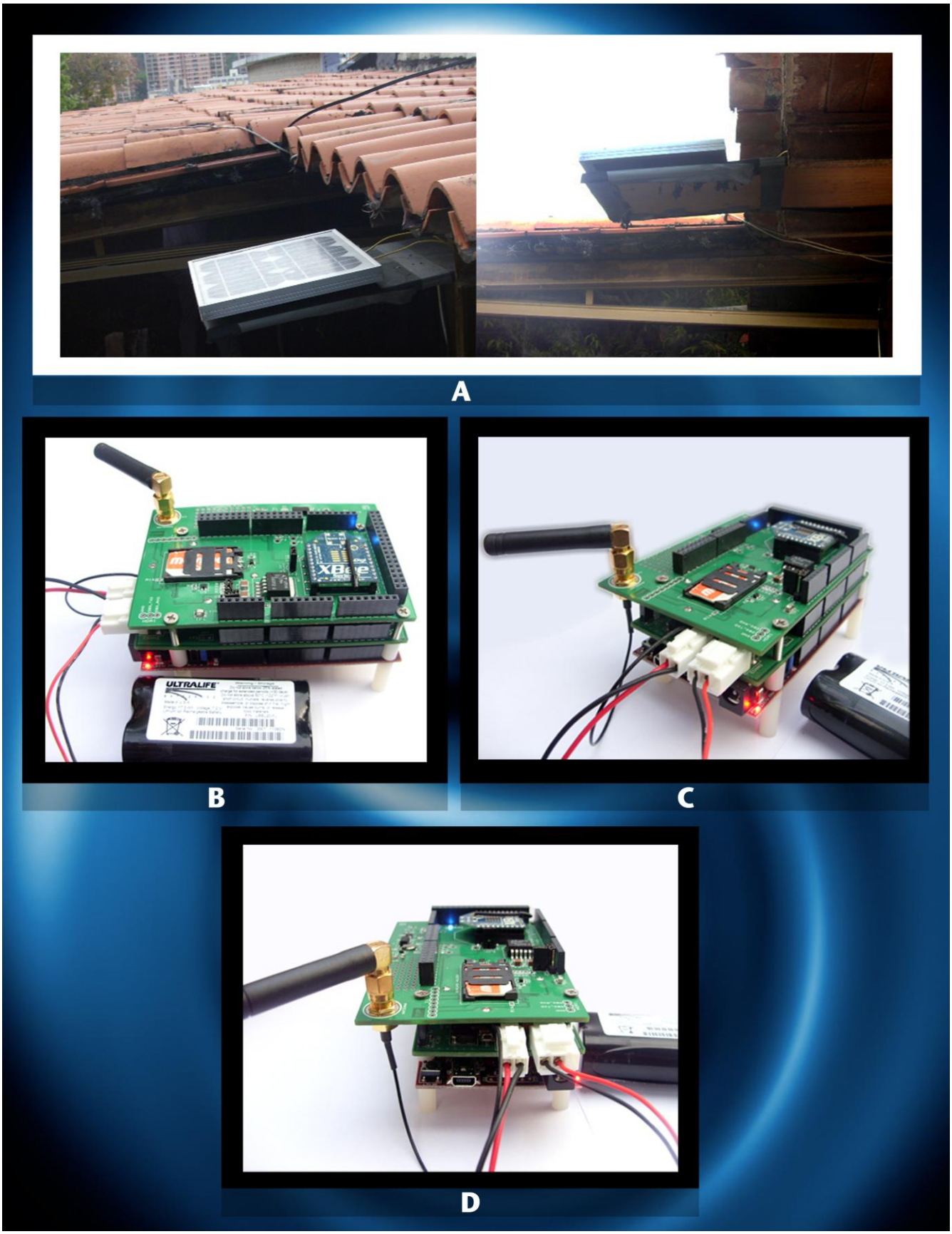


Fig. 1 The picture of the complete assembly.

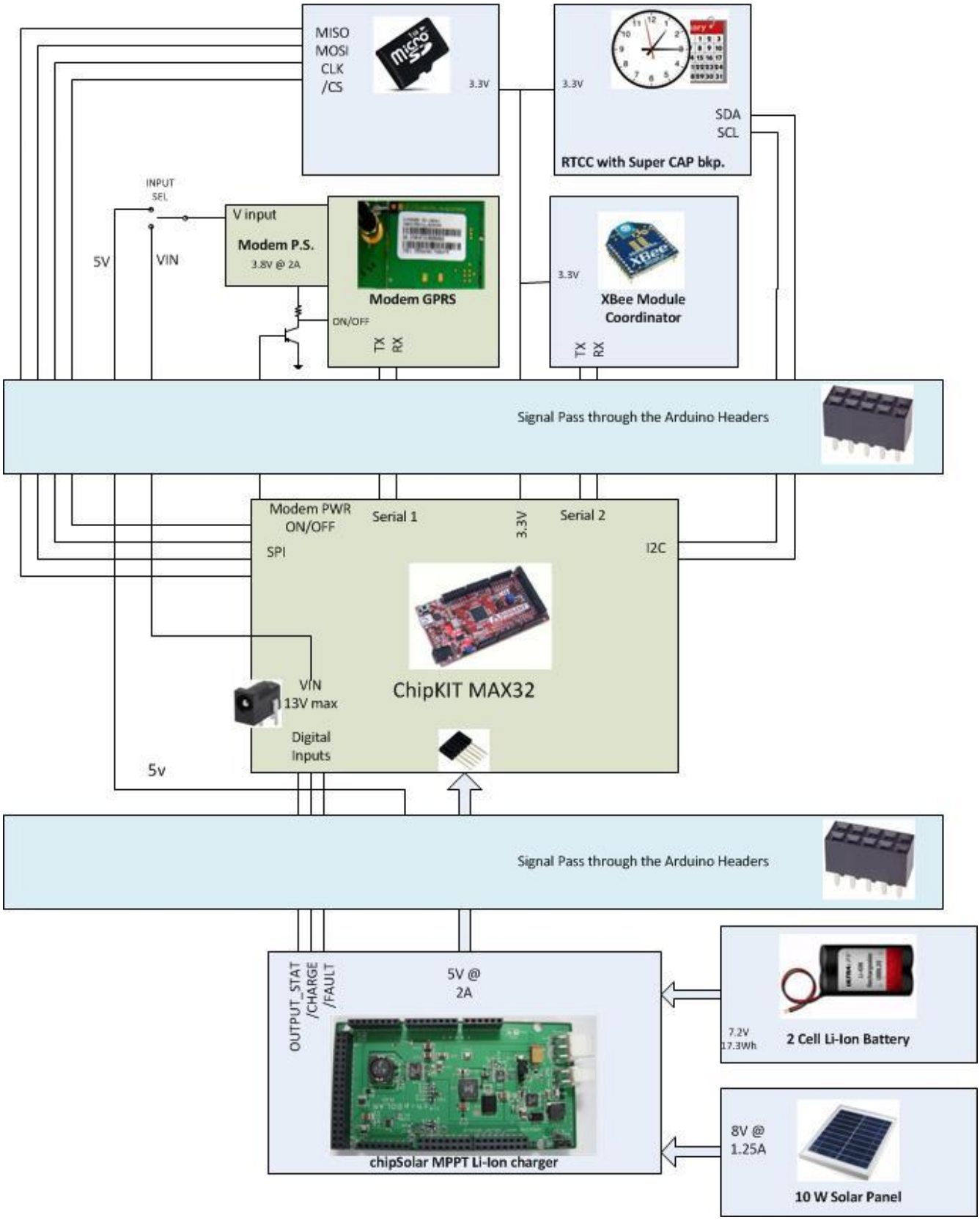
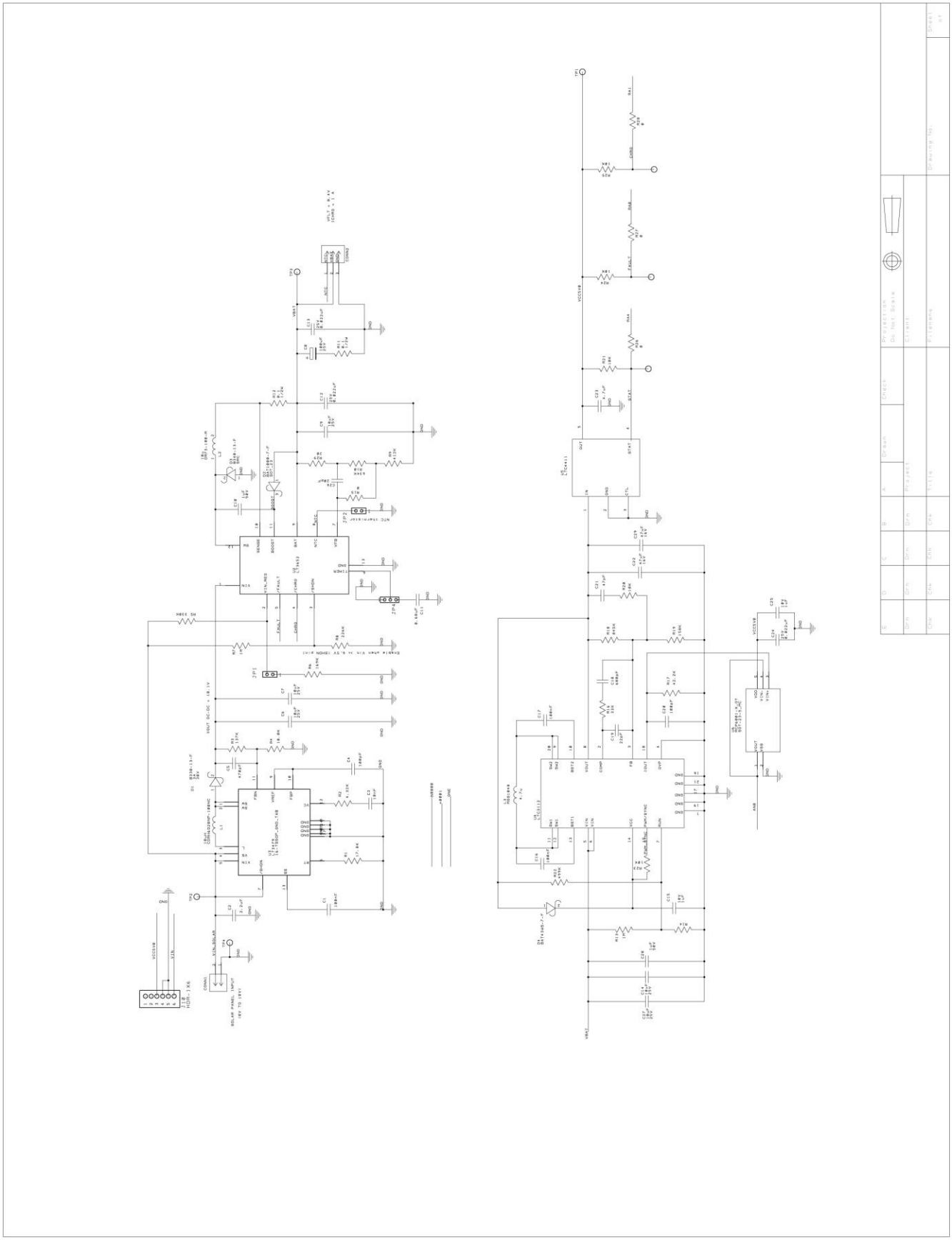


Fig. 2 Block Diagram of the built system.



g # 3 . chipSOLAR schematic.

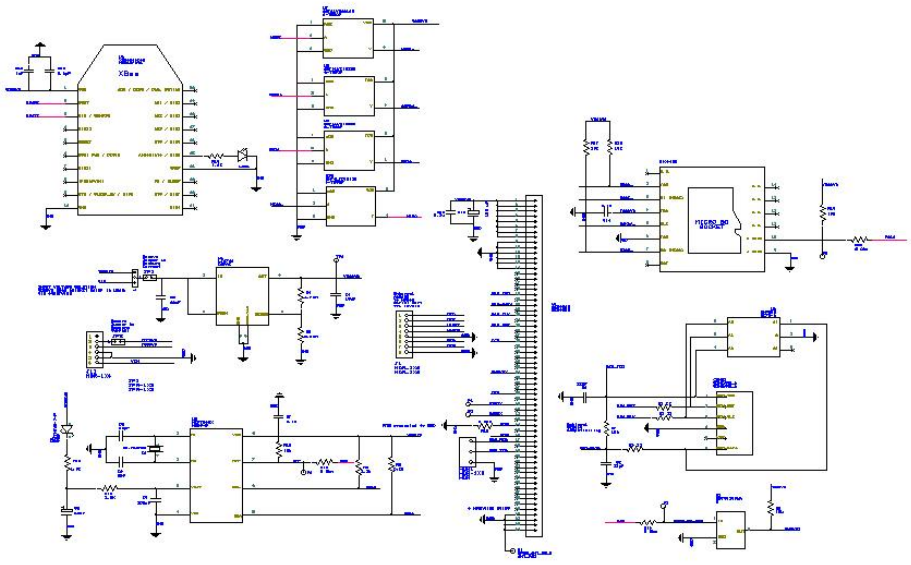


Fig. 4. chipWireless schematic. Please refer to source files for details.

Next there is some part of the Main Code. Please refer to the complete documentation for details.

```
#include "system.h"
#include <QueueList.h>
#include <Time.h>
#include <Wire.h>
#include <MCP794xxRTC.h>    // a basic MCP794xx library
#include <EEPROM.h>

extern MCP794xxRTC RTC;// = MCP794xxRTC(2); // create an instance of device MCP79402

const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables for the chipSolar board
int V5_status;
int charger;
int fault;

int buttonState, buttonStateLast;         // variable for reading the pushbutton status and hold the last value
String pvalue;                            // pachube value of the form <stream_id>,<value>
String IP, data;
char buffer[255];
char temp;                                // these variables are used for the reception and processing of Serial PC commands.
char tph[15];
String tempCmd, tempPh;
int addr;
extern uint8_t entry_num;                 // this is the number of entries of phone numbers we have in memory
extern uint8_t temp_addr;                 // this is the last available address to save the next entry, one byte will give us more
or less than 20 entries depending on ph number length
extern uint8_t addr_last;                 // this is the eeprom address variable we will use to save the phone numbers...
int j;
int query_entry;

void setup()
{
  // Digital I/O initialization
  pinMode(MODEM_PWR, OUTPUT);
  digitalWrite(MODEM_PWR, LOW);         // a High Level applied at the PWRKEY Pin of Modem (remember the NPN
transistor)
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the switch button pin as an input:
  pinMode(buttonPin, INPUT);
  #if not defined __bttn_sim__
  buttonState = digitalRead(buttonPin);
  #else
  buttonState = 0;
  #endif
  buttonStateLast = buttonState;
  pinMode(LT4411_STAT, INPUT);
  V5_status = digitalRead(LT4411_STAT);
  pinMode(LTC3652_CHRG, INPUT);
  charger = digitalRead(LTC3652_CHRG);
  pinMode(LTC3652_FAULT, INPUT);
  fault = digitalRead(LTC3652_FAULT);

  // start serial port PC-chipKit comm at 9600
```

```
  Serial.begin(9600);
  Serial.println("a>PC console is up!");
  // start serial port at 57600 bps
  Serial1.begin(115200);

  delay(1000);
  Serial.println("a>Send something to start");
  Serial.flush();
  while(!Serial.available());
  delay(200);                    // allow a bunch of characters to be discarded.
  Serial.flush();
  // wait for serial monitor to send anything... just for debug

  Serial.println("a>Call to modemInit");
  while(!modemIsInitialized()){
    modemInit("GSM850_PCS_MODE");      // some basic initialization and test communication with modem
    delay(500);                    // if we should retray wait some time
  }

  Serial.println("a>Call to modemCheckNetwork");
  while(!modemIsRegistered()){
    modemCheckNetwork();
    delay(1000);
  }
  Serial.println("a>Call to modemInitGPRS");
  if(!modemInitGPRS("IP", "int.movilnet.com.ve")){
    while(1);                      // Error while defining PDP context or other GPRS parameter.
  }

  Serial.println("a>Call to modemActivateGPRS");
  #if defined __SIM340__
  modemActivateGPRS("int.movilnet.com.ve", 1);         // Multi-Ip is not supported for this modem, anyway I start working
with this modem and they release the SIM900 series
  #else                                  // see modemActivateGPRS functions for detail, the multi commands should be apply
always to fully configure module.
  modemActivateGPRS("int.movilnet.com.ve", 0);         // which allow you to do multi-ip, set the second parameter to 1 in
order to allow it.
  #endif

  Serial.println("a>Call to modemSMS_init");
  modemSMS_init();

  // get some state and working variables
  RTC.read_byte(eeprom_phone_entry_num, entry_num);
  RTC.read_byte(eeprom_phone_entry_last, addr_last);
  #if defined __sram_wr_debug__
  Serial.print("eeprom_phone_entry_last value is= ");
  Serial.println(addr_last, HEX);
  Serial.print("eeprom_phone_entry_num value is= ");
  Serial.println(entry_num, HEX);
  #endif

  // Initialize the ZigBee Network
  InitSensorNetwork();
  Serial.println("a>setup is done!");
  // End of setup
}
```