

THE GADGET FREAK FILES CASE #168

Gas Sensors Sniff Out Danger

Natanel Eizenberg decided to build a sensor module that would detect and measure carbon monoxide and methane to help ensure safe environments for his coworkers and family. But in addition to creating an alarm circuit, Natanel added a wireless link that communicates with a nearby PC. Adding a wireless gateway could extend monitoring and control operations to the Internet.

In this Gadget Freak project, an Atmel AVR32 microcontroller (MCU) monitors signals from a carbon-monoxide sensor, a methane sensor, and a temperature sensor. The MCU double buffers the results from the MCU's analog-to-digital converter (ADC) for the three sensors. Double buffering requires two sections of memory. As the software fills one buffer with sensor data, a second buffer--filled previously--sends its data to a host computer or to an XBee transceiver.

When the transmission ends and the MCU has filled the other buffer, the roles of the buffers switch. The just-filled buffer provides data for transmission while the MCU fills the buffer previously used to hold data. The memory-read and memory-write switch back and forth between the buffers.

Editor's Note: The AVR32 MCU does not provide a specific "double-buffer memory." Programmers implement this type of operation in their code, as explained later.

[Download a ZIP file of source code, schematic diagram, and MATLAB files from: www.gfreak.com/GF167/GF167.zip.](http://www.gfreak.com/GF167/GF167.zip)

A Digi International XBee wireless module transmits the information upon request to a nearby monitoring station, usually a PC. The buffer for the ADC data relies on a data structure in the code that saves three values in each buffer "slot." Those 11 slots each hold a carbon monoxide, a methane and a temperature value. Setting the buffer to 11 slots for this type of information reduces wireless traffic because a single transmission can transfer a large block of buffered data. The wireless communications relied on Digi International 2.4 GHz XBee ZB (ZigBee) modules. Digi provides a variety of module types with different types of antennas and ranges. Programming, though, remains the same from one module type to another type.

The AVR32 MCU module can operate as a stand-alone device (without transmitting data). When it detects carbon monoxide or methane, algorithms based on thresholds and a time window can determine whether to activate LEDs, a piezoelectric buzzer, and possibly other devices connected to general-purpose I/O (GPIO) pins on the MCU. You might have a GPIO pin turn on a ventilator, for example.

Project Description

Figure 1 shows the two boards used in the gas-detector circuit. The left-most board holds the analog parts and a socket for an XBee wireless module. The pre-built board on the right provides an AVR32 AT32UC3A1512 MCU and its support components. You can buy this board from Alvidi (Frankfurt am Main, Germany). Part number: AL-70, cost: approx. 30 euros. For more information, go to: http://alvidi.de/avr32_module.html.

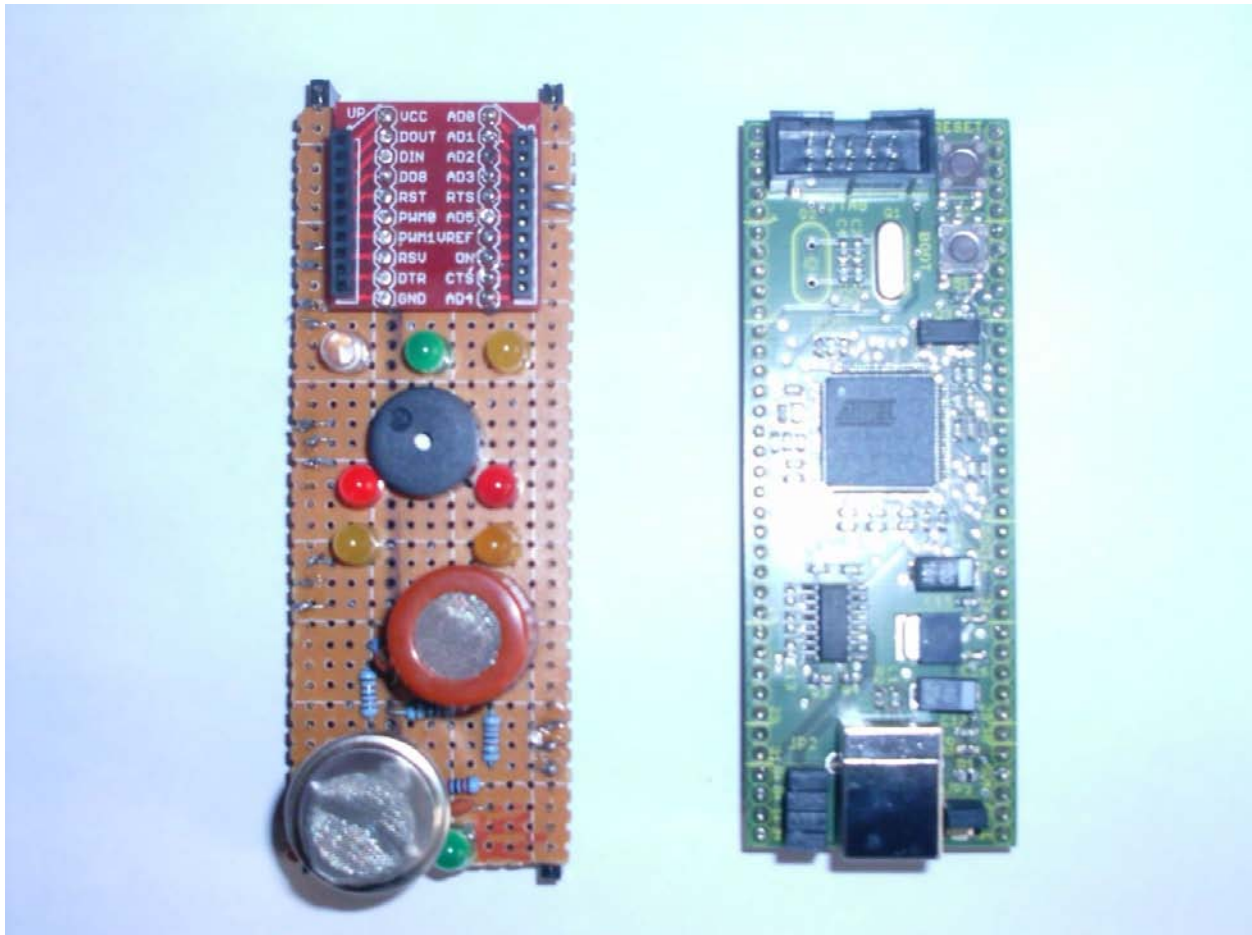


Figure 1. The sensor board with XBee socket (left) and the digital board with AVR32 MCU (right).

The two boards connect back to back, as shown in **Figure 2**. The assembled boards obtain 5V power through the USB connector on the AVR32 board. If you use an XBee transceiver module, a USB cable supplies only power. If you do not need a wireless link, you do not need an XBee transceiver and the AVR32 MCU can communicate with a host PC via the USB cable and a virtual serial port. When used as a stand-alone sensor with a wireless link you can supply 5V power from a plug-in wall adapter or other power supply.

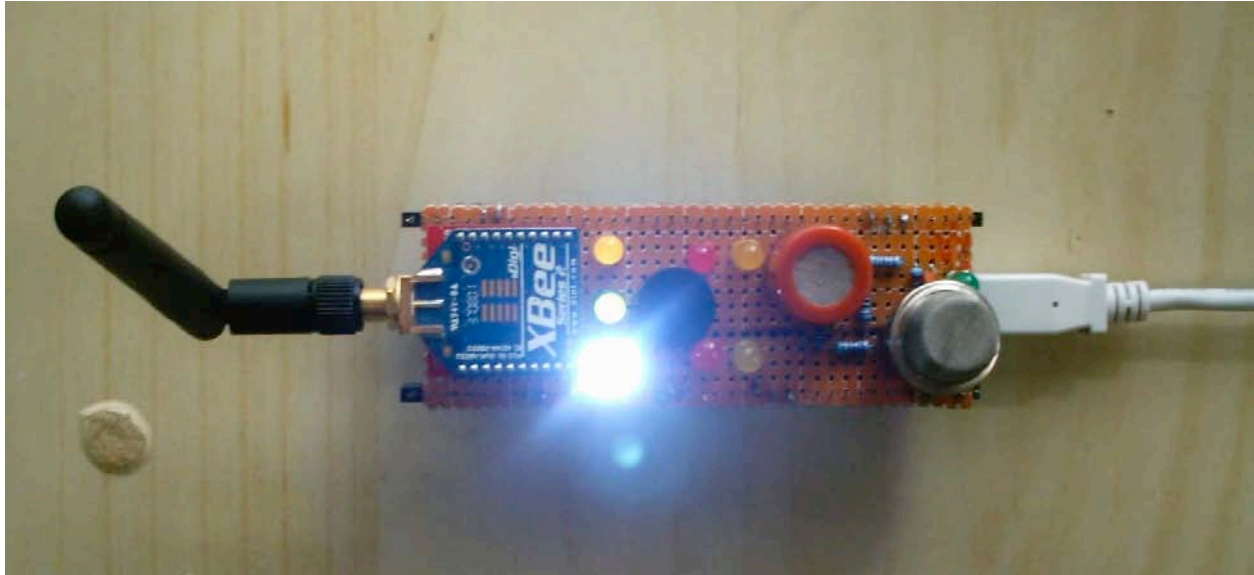


Figure 2: The sensor board attaches to the AVR32 MCU board in piggy-back fashion.

Every 100 milliseconds the MCU samples the analog sensors, performs an analog-to-digital conversion, and stores the results. Separate tasks run the data-analysis algorithm and transmit the resulting data when a buffer is full. A simple graphical user interface (GUI) created with MATLAB show gas and temperature measurements and device messages.

Analog Board

The analog board shown in **Figure 3** holds the three sensors (two gas sensors and a thermistor), LEDs, a buzzer, and a socket for an XBee transceiver module. The two analog gas sensors, labeled MQ-4 (methane) and MQ-7 (carbon monoxide) in the accompanying schematic diagram, receive 5V power continuously to maintain the sensor elements at their working temperature. Thus, you should not attempt to power the sensor and MCU circuit solely from a battery. You could use a battery for backup power, however.

The two gas sensors use resistors on their output to reduce the sensor output from 0-5V to 0-3.3V to provide signals compatible with the MCU's ADC. Each sensor has two LEDs associated with it, although there's no direct connection between sensors and LEDs.

The original circuit does not provide a reset pushbutton but you could include a normally closed pushbutton on the power input to the circuit to reset the alarms. Also, you can use the GUI to "close" monitoring of individual or all ADC channels. Just enable or disable a channel (small square mark) in the GUI. This action sends an enable/disable parameter to the MCU.

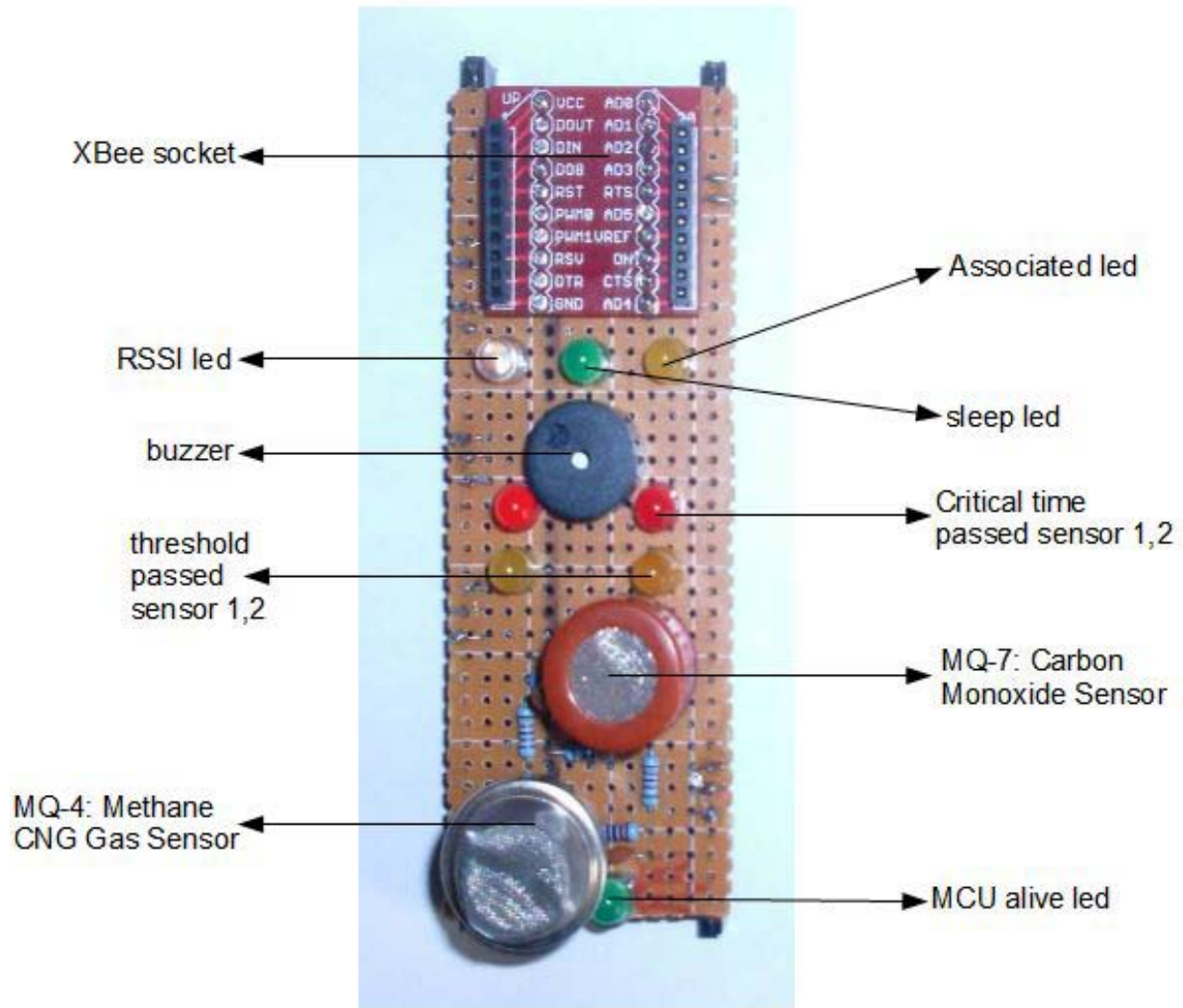


Figure 3: Analog board components.

An "MCU alive" LED flashes every second to indicate that the MCU is still running. Three LEDs connect to the XBee module to indicate Associated and Sleep conditions, and signal strength (RSSI: received signal strength indicator). The Associated LED indicates the XBee can communicate with a remote XBee device. The Sleep LED is not used because Natanel did not put the XBee module into a low-power mode. (The XBee module on the sensor unit serves as a router and not as an end device.) Natanel created the analog board with the same size as the AVR32 MCU board so he could connect them easily.

Editor's Note: Natanel explained that power to the heater internal to the MQ-7 carbon monoxide sensor should run through a 150-second cycle: 5V for 60 seconds followed by 1.4V for 90 seconds. The original circuit kept the heater on constantly at 5V. For more information, see the sensor data sheet at:

<http://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf>. You could use a PWM output to drive a transistor that would switch the heater to ground (with a 5V supply) and then adjust the PWM duty cycle to decrease the average voltage across the heater to 1.4V. The heater element draws about 150 mA at 5V and about 40 mA at 1.4V. You might need a capacitor across the transistor to filter the PWM signal.

See **Figure 4** for a complete schematic diagram of the sensor board. Find information about the AVR32 board at the Alvidi web site.

Editor's Note: The schematic diagram and the Bill of Materials lists a 130 KΩ resistor. Connect a 120 KΩ and a 10 KΩ resistor--both listed in the BOM--in series to produce the needed 130 KΩ resistance.

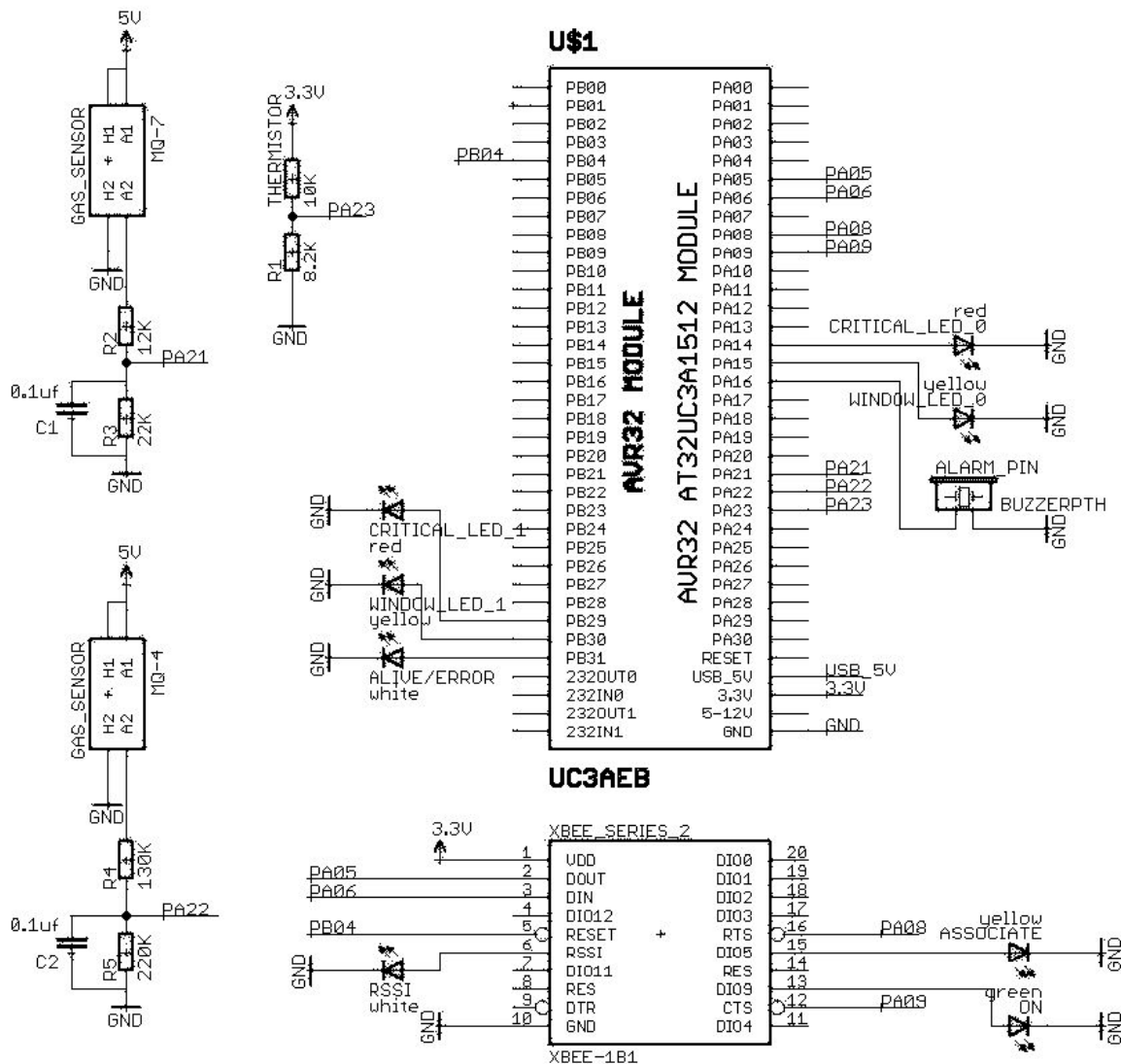


Figure 4. Analog circuit schematic diagram.

Digital Board

The digital board comes from Alvidi, as noted earlier. For information about the AVR32 MCU, part number T32UC3A1512, go to: www.atmel.com/products/avr/default.asp. Natanel chose this MCU because it provided more than enough flash memory and RAM for this project and for future projects that would use the same code.

The MCU board received 5V power through the USB connector to a host computer. An on-board regulator reduce the power to 3.3V for the MCU, the analog circuits, and the XBee transceiver.

Algorithms

The three main software tasks in this project run under the FreeRTOS operating system, as shown in **Figure 5**. For FreeRTOS information, visit: www.freertos.org.

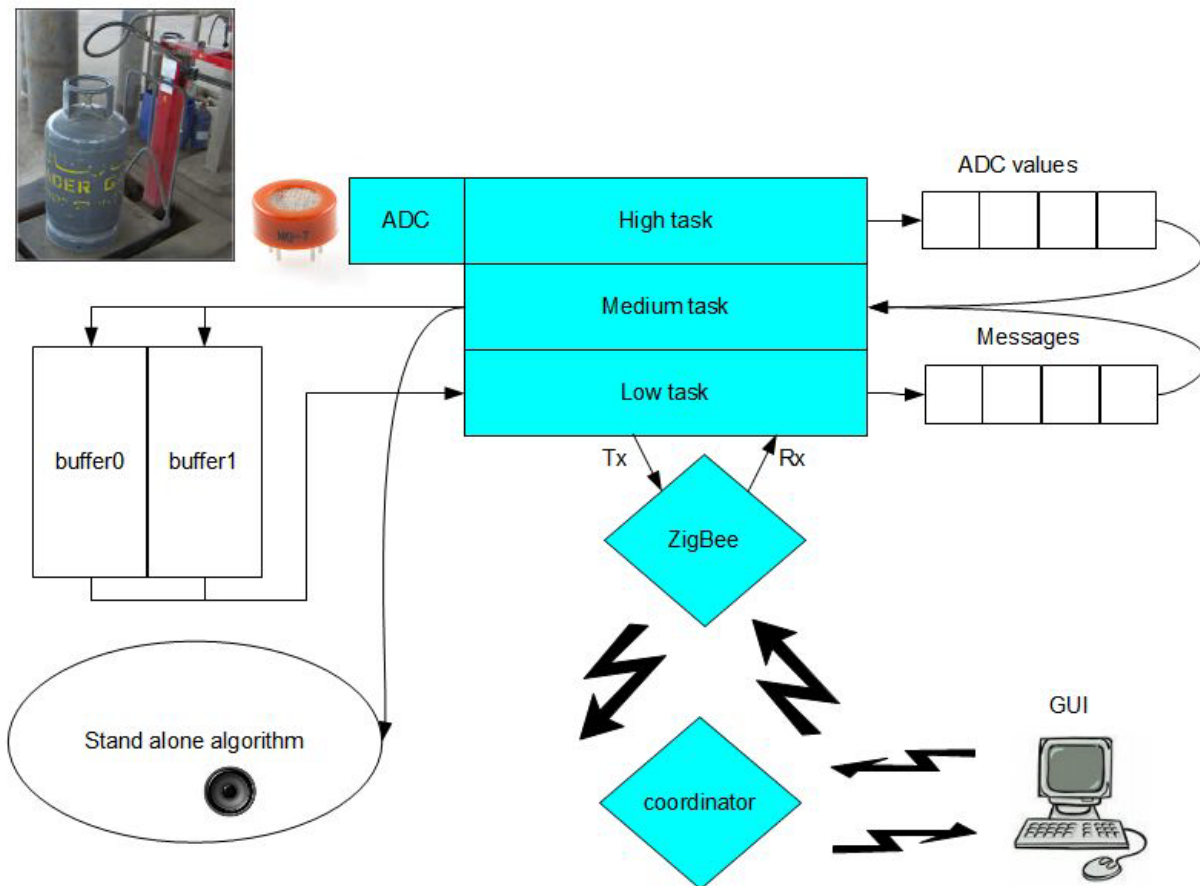


Figure 5: Software block diagram.

Task #1: High task

The high task reads values from the MCU's ADC, copies new parameters (if any) from temporary "shadow" storage into the working-parameter area, and handles any error conditions.

a. The ADC measurements occur every 100 milliseconds and the MCU stores 11 values in the double-buffer area of memory. (The Medium task handles those values. In case the medium task requires a longer-than-expected time, the double buffer continues to store new ADC values for later handling.)

b. The Low tasks can receive new parameters via wireless communications from a remote PC or via a USB cable. Any new parameters received go into temporary shadow storage and then the High task immediately copies them to replace older parameters. The MCU also saves these new parameters in flash memory so if you reset the MCU it immediately obtains the latest parameters the module received.

c. Errors checks fall into two categories:

1. Critical errors, which include task-switch error (such as tasks taking too long) and queue overrun. When the MCU detects a critical error it changes the flashing Alive LED to constant-on, writes a critical-error message into flash memory, and enters an infinite loop. When you next reset the module, the MCU reads the critical message and transmits it. (You can include other ways to handle and report errors.)

2. Non-Critical errors; which include buffer over runs. This condition happens when one buffer (see Figure 5) has not transmitted its data and the software needs to use that buffer for new ADC data. A message over run, which occurs when a message does not have any more space in memory, also triggers a non-critical error. This type of error causes the MCU to transmit an error message and to store that message in flash memory. But, the MCU continues to work normally.

Task #2: Medium task

The medium task takes the ADC data from the ADC queue and places it in the appropriate buffer, stores message data from the message queue to the double buffer and processes the ADC values.

The medium task reads the ADC values and messages from their associated queues and puts them in the next available space in the buffer, as shown in **Figure 6**. The buffer has a maximum "depth," noted as MAX_LENGTH or Buffer max size. The first section, ADC AREA, has a fixed length and stores sensor data as described earlier. The second section, MESSAGE AREA, has a varying size that increases or decreases with the length of messages collected from the software and ready for transmission.

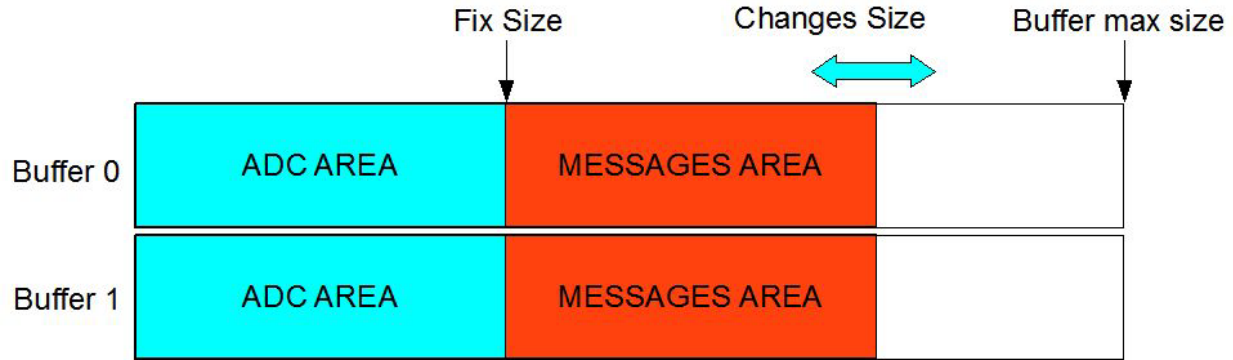


Figure 6: Diagram of buffer-memory space. The ADC area has a fixed size while the message area can expand and contract as needed.

The software transmits only the information in the ADC AREA and the filled MESSAGE AREA. Thus, a transmission does not normally extend to the buffer's maximum size. The `MAX_LENGTH` of the buffer amounts to 83 bytes and the examples below show the number of transmitted bytes for four situations:

1. buffer 0: 33 bytes of data + one message (10 bytes) → transmit 43 bytes
2. buffer 1: 33 bytes of data + no messages → transmit 33 bytes
3. buffer 0: 33 bytes of data + four messages (40 bytes) → transmit 73 bytes
4. buffer 0: 33 bytes of data + no messages → transmit 33 bytes

When the MCU fills a fixed-size ADC buffer, it "marks" the buffer as ready to transmit. If the medium task receives a data request from a user, it transmits the information in the full, or ready-to-transmit, buffer.

The algorithm created by Natanel works specifically with the gas-sensor information and it can operate with or without an XBee transceiver. You can change the algorithm to suit your sensors or requirements. Although the algorithm applies to both gas sensors, the data from each sensor remains separate as do the threshold and time-window parameters.

Figure 7 shows a sensor's response to a gas as a curved line that shows an increasing and then a decreasing gas concentration. The x axis plots time and the y axis represents a sensor value.

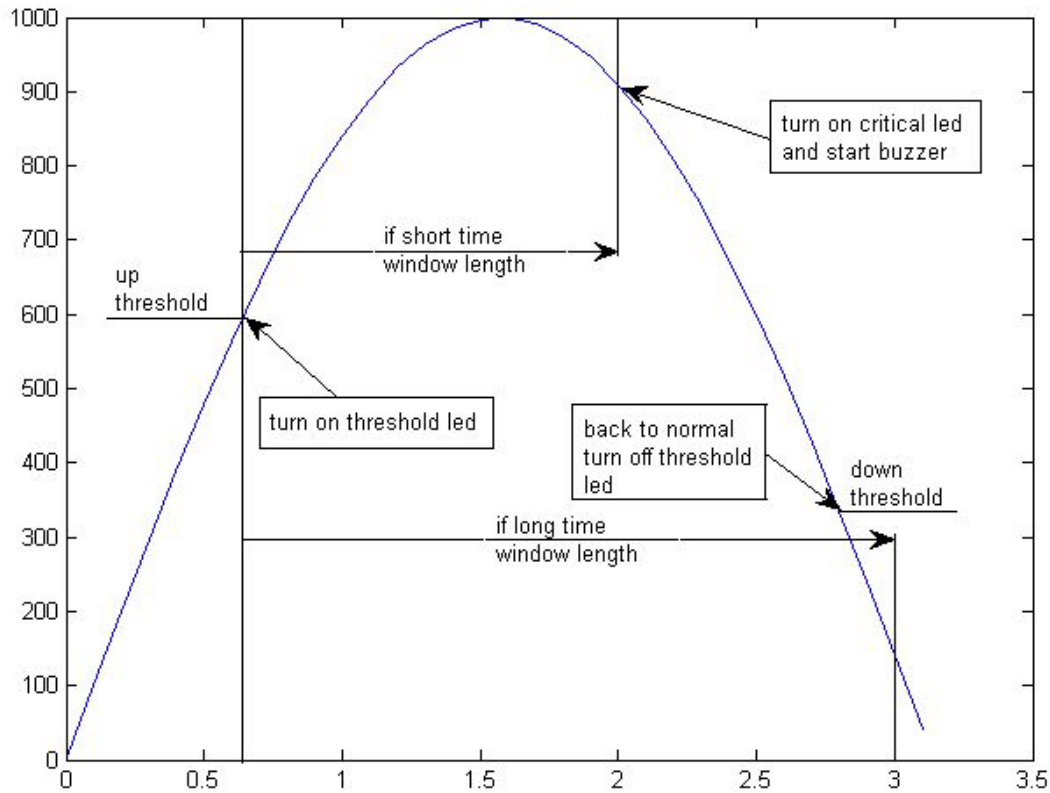


Figure 7. Diagram of how the sensor-alarm algorithm works.

When the sensor value remains below the "up threshold" value, there is no hazard present.

When the sensor value exceeds the "up threshold" the MCU turns on the "threshold-passed LED" and starts a count. Assume you chose a short time for the counter. If the MCU determines the gas concentration exceeds the "up threshold" value at the end of the count, it turns on the "critical LED" and the audible buzzer.

When the sensor value drops below the "down threshold," the MCU turns off the buzzer, the "critical LED," and the "threshold LED."

Now assume you incorrectly chose a long time for the counter. At the end of the count, the MCU measures the gas concentration below the "down threshold," so the "critical LED" never turns on and the buzzer never sounds. You must choose a count value for your situation. Or, you could choose a very short count and simply turn on the "critical LED" and buzzer close to the "up threshold" point.

Task #1: Low task

The low task flashes the "alive LED" and handles transmission and reception of information to and from the MCU and the XBee transceiver. The GUI polls the MCU to obtain information. The GUI send a "request data" command to the XBee module attached to the PC. The request reaches the XBee module and then the MCU on the sensor board. If the MCU has information in a buffer memory, it transmits the number of bytes in the transmission, followed by the information in the buffer memory. If the MCU has no buffered data--including no messages--it transmits only a zero (0) to indicate no data.

Users can enter the following commands via the GUI, which the "low task" will handle:

PARAMS_IN, a user command that requests parameters values from MCU, this will be followed by the transmission of parameters from MCU to GUI.

PARAMS_OUT, a user command that requests an update of parameters, this will be followed by the parameters sent from the GUI to the MCU.

PARAMS_RESET, a user command that requests a reset of parameters based on those provided in the code for the project. When you reset the parameters with this command, the MCU reverts to the original hard-coded values and it also writes them as parameters in flash memory.

DATA_IN command, a user command that requests transmission of data from the buffer that next fills. The MCU transmits the full-buffer values to the GUI.

DEBUG command is for task debug, it will send tasks entering and exiting time array, as described below:

General

Natanel created his software to make it easy for another programmer to change parameters in the #define lines to accommodate other AVR32 processors and boards. The list below describes some of these features:

- Telemetry – sending or receiving using XBee or USB.
- Sampling time – sampling time including time rate of each task.
- ADC bit size – sampling at 8 or 10 bits (8-bit values reduce buffer size and transmission time).
- ADC Buffer size – number of ADC values in a buffer which also determines the time for a full-buffer transmission.
- Buffer message size – size of max available space for messages in buffer.
- Number of channels to sample – from 1-8 sampling channels.
- Power saving – enable/disable LEDs, enable/disable buzzer, entering sleep mode in idle task.
- USART debug modes – USART debug of telemetry messages, tasks duration, errors, parameters changing. Initialized procedure is shown in **Figure 8**.

- Other debug capabilities such as simulated input signals (sine, cosine, and others), display tasks switch, and so on.

```

COM1 - PuTTY
MMMMMMMMM  NNNN  NNNNN  A  TTTTTTTTTTTTTTTT
MM  .MM  NNNNN.  .N  AA  TT  TTTTTT  TT
MMM  .M  .NNNNNNN  .N  AAA  T  TTTTTT  T
MMMMM  .N  NNNNNNN  .N  AAAAA  TTTTTT
MMMMMMMM  .N  NNNNNN  .N  AA  AAAA  TTTTTT
MMMMMMMMM  .N.  .NNNNN  .N  AA  AAAAA  TTTTTT
MMMMMMMMM  .N.  .NNNNNN  .N  AA  AAAAAA  TTTTTT
MMMMMMMMM  .N.  .NNNNNNN  .N  AA  AAAAAA  TTTTTT
MMMMM  .N.  .NNNNN  .N  AAAAAAAAAAAAAA  TTTTTT
M  .MM  .N.  .NNNN  .N  AAA  AAAAAAA  TTTTTT
MM  .MM  .N.  .NNN  .N  AAA  AAAAAAA  TTTTTT
MMMMMMMMMMMM  .N  .NN  .N  AAAAA  AAAAAAA  TTTTTT
GCC 4.3.2 Jan 20 2010 19:15:33 NATANEL_EIZENBERG_AVR32_UC3_972_54_5726016

-----user defined-----
..\src\APPLICATION\HW_init.c:656:
..\src\APPLICATION\HW_init.c:657: CLK_CPU = 48000 KHz
..\src\APPLICATION\HW_init.c:658: CLK_FBA = 24000 KHz
..\src\APPLICATION\HW_init.c:659: CLK_OSC0 = 12000 KHz
..\src\APPLICATION\HW_init.c:660: ADC number of sampling bits=10
..\src\APPLICATION\HW_init.c:661: ALG_VER=1
..\src\APPLICATION\HW_init.c:662: USB_TELEMETRY=DISABLED
..\src\APPLICATION\HW_init.c:663: ZIGBEE_TELEMETRY=ENABLED
..\src\APPLICATION\HW_init.c:664: configTICK_RATE_HZ=50
..\src\APPLICATION\HW_init.c:665: ONE_TICK_TIMERATE in msec=20
..\src\APPLICATION\HW_init.c:666: ONE_SEC_TIMERATE in ticks=50
..\src\APPLICATION\HW_init.c:667: ONE_MIN_TIMERATE in ticks=3000
..\src\APPLICATION\HW_init.c:668: SAMPLING_TIME_MSEC=100
..\src\APPLICATION\HW_init.c:669: SAMPLING_TIME_TICK=5
..\src\APPLICATION\HW_init.c:670: HIGH_TASK_TIMERATE TICK=5
..\src\APPLICATION\HW_init.c:671: MEDIUM_TASK_TIMERATE TICK=5
..\src\APPLICATION\HW_init.c:672: LOW_TASK_TIMERATE TICK=20
..\src\APPLICATION\HW_init.c:673: USB_TASK_TIMERATE TICK=1
..\src\APPLICATION\HW_init.c:674: ADC_SAMPLING_TIMERATE TICK=5
..\src\APPLICATION\HW_init.c:675: GPIO_TIMERATE TICK=5
..\src\APPLICATION\HW_init.c:676: ALG_1_TIMERATE TICK=5
..\src\APPLICATION\HW_init.c:677: TELEMETRY_TIMERATE TICK=20
..\src\APPLICATION\HW_init.c:678: USART_TIMERATE TICK=20
..\src\APPLICATION\HW_init.c:679: USB_TIMERATE TICK=20
..\src\APPLICATION\HW_init.c:680: CRITICAL_ERROR_CHECK_TIMERATE TICK=10
..\src\APPLICATION\HW_init.c:681: NONE_CRITICAL_ERROR_CHECK_TIMERATE TICK=50
..\src\APPLICATION\HW_init.c:682: NUM_OF_ADC_SAMPLING=20
..\src\APPLICATION\HW_init.c:683: ADC_NUM_OF_CHANNELS=3
..\src\APPLICATION\HW_init.c:688: MAX_GPIO_INPUT_PINS=NONE
..\src\APPLICATION\HW_init.c:694: MAX_GPIO_OUTPUT_PINS=NONE
..\src\APPLICATION\HW_init.c:696: MESSAGE_BUFFER_SIZE_BYTES=128
..\src\APPLICATION\HW_init.c:698: ADC_SIZE=12 (bytes), ADC_QUEUE_SIZE=96 (bytes)
..\src\APPLICATION\HW_init.c:699: MESSAGE_SIZE=24 (bytes), MESSAGE_QUEUE_SIZE=192 (bytes)
..\src\APPLICATION\HW_init.c:700:
..\src\APPLICATION\HW_init.c:347: -----ADC buffers-----
..\src\APPLICATION\HW_init.c:379: #BufIndex[0]=SIN
..\src\APPLICATION\HW_init.c:390: #BufIndex[1]=COS
..\src\APPLICATION\HW_init.c:400: #BufIndex[2]=DCOMB
..\src\APPLICATION\HW_init.c:355: #BufIndex[3]=CHANNEL_OFF
..\src\APPLICATION\HW_init.c:355: #BufIndex[4]=CHANNEL_OFF
..\src\APPLICATION\HW_init.c:355: #BufIndex[5]=CHANNEL_OFF
..\src\APPLICATION\HW_init.c:355: #BufIndex[6]=CHANNEL_OFF
..\src\APPLICATION\HW_init.c:355: #BufIndex[7]=CHANNEL_OFF
..\src\APPLICATION\HW_init.c:493:
..\src\main.c:25: passed initialization of all HW
..\src\TASK_SCHEDULER\TaskScheduler.c:102: -----program checks/initializations-----

```

Figure 8: This type of information appears on a host PC during initialization of the MCU.

The Universal Synchronous/Asynchronous Receiver Transmitter (USART) connects the MCU to the host computer that displays debug information, system details, and MCU-configuration information when you start the sensor module.

The initialization screen displays:

1. General user-control information: MCU clock frequency, clock-tick time of each task, in-task function clock-tick times, USB-enabled, XBee transceiver enabled, GPIO functions, buffer size, etc.
2. ADC buffer setting, simulator or real data, and number of ADCs connected.
3. Error status at reset, parameters copied from flash memory at initialization, and parameter values.

The graphical user interface (GUI)

Natanel wrote a simple GUI in MATLAB, which uses serial commands to communicate with the XBee connected to the PC. You can use the GUI to control many portions of the sensor module, as shown in **Figure 9**:

- start/stop button
- arrived buffers counter
- task debug button
- set button to send parameters changes (from GUI to device)
- reset max time counter
- time counter of last read buffer
- max time of read buffers
- ADC values running scroll plot
- available channels numbers in blue
- algorithm enable/disable for each channel using check box
- up thresholds
- down thresholds
- time window in milliseconds
- arrived message window

Editor's Note: Design News provides Gadget Freak circuits and designs for educational and experimental purposes and not for use in production or safety-related applications. Builders assume all responsibilities for the use or adaptation of information provided in or for a Design News Gadget Freak.

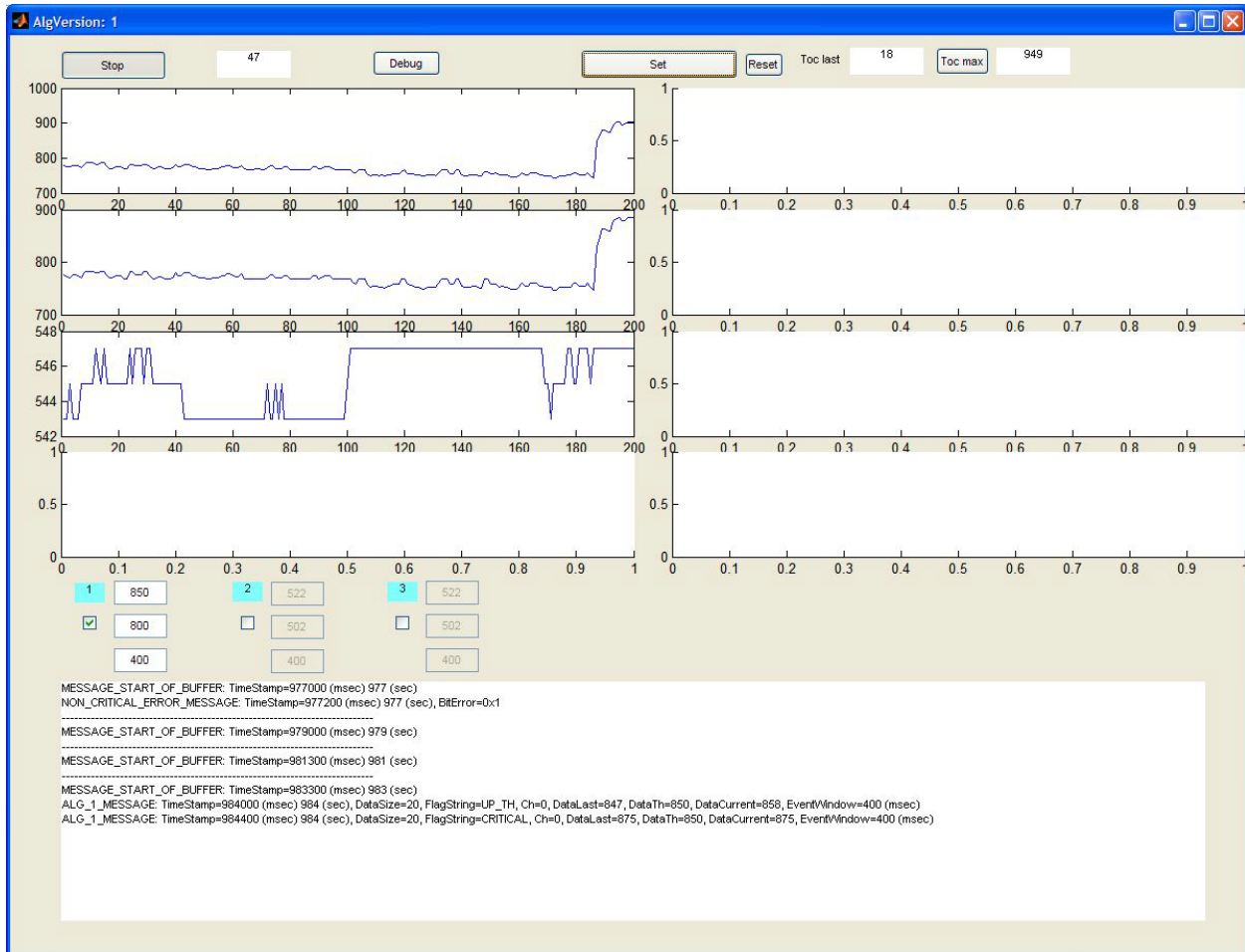


Figure 9: MATLAB graphical user interface (GUI).

The user can change each of the parameters, after which the GUI "arms" the Set button and changes its color to red. Pressing it sends these new parameters to the MCU and clears the button's red color. After you press the Start button, the GUI periodically sends a buffer-request command to the MCU that will reply with data as soon as it finds a full buffer.

MATLAB uses simple serial-communication commands to connect with the PC's XBee module, so you can substitute a simple serial-communication emulator or a program that provides serial-port control. You could use an XBee USB Adapter Board from Parallax (part no. 32400, \$US 19.95) and the free X-CTU software from Digi to communicate with a PC-based XBee module. (www.parallax.com) Or, use Visual Basic or another language that can control a PC-based serial port or USB virtual serial port. It's up to you to create a GUI for these substitute programs if you need such a display.

Follow the proper protocol to send commands. (The source-code file `telemetry.c` can help as can the details of code shown below). Natanel did not write a full API for this

project. If you need assistance, contact Natanel directly by email at: jn_eizenberg@hotmail.com. Type: "GF167--Gas Sensor" in your email subject line.

```
// command messages
#define NONE 0
#define PARAMS_IN 1
#define PARAMS_RESET 2
#define PARAMS_OUT 3
#define DATA_IN 4
#define DATA_ECHO 5
#define DEBUG 6
```

```
typedef struct COMMAND_MESSAGE
{
    unsigned portCHAR ucCommandType;
    portSHORT sValue1;
    portSHORT sValue2;
    portSHORT sValue3;
} xCommandMessage;
```

PARAMS_IN

sCommandMessage.sValue1 = which params set is requested for broadcasting HW_PARAMS,ALG1_PARAMS, etc. see parameters.h file.

Following broadcasting requested structs of parameters

PARAMS_RESET

sCommandMessage.sValue1 = which params set is requested to reset

PARAMS_OUT

sCommandMessage.sValue1 = which params set is going to come HW_PARAMS,ALG1_PARAMS,etc. see parameters.h file.

Following structs of parameters see parameters.h file.

DATA_IN

sCommandMessage.sValue1 = none

Following transmitting of data length and the data.

DEBUG

sCommandMessage.sValue1 = none

following transmit of special task switch buffer recorder (defined in code) cDebugBuff[DEBUG_BUF_LENGTH]

Bill of Materials

Amt	Part Description	Allied Part #	
2	0.1 μ F Capacitor, 50V	852-1170	
1	10K Ω Resistor 1/4W	296-4743	
1	8.2K Ω Resistor 1/4W	296-6598	
1	10K Ω Resistor 1/4W	296-4743	
1	12K Ω Resistor 1/4W	296-6491	
1	22K Ω Resistor 1/4W	296-4755	
1	120K Ω Resistor 1/4W	296-6490	
1	130K Ω Resistor 1/4W	<i>See Text</i>	
1	220K Ω Resistor 1/4W	296-4757	
1	LED, High Intensity, White	749-3330	
1	Vector Electronics Perf-board	977-1250	
1	Mallory Sonalert buzzer	854-0119	
Amt	Part Description	Part #	Source
1	ALVIDI AVR32-Module	AL-UC3AEB	Alvidi
1	MQ-4 methane compressed natural gas (CNG) gas sensor	SEN-09404	SparkFun Electronics
1	MQ-7 carbon monoxide sensor	SEN-9403	SparkFun Electronics
1	10K thermistor	SEN-00250	SparkFun Electronics
1	XBee XBP24-BSIT-004	WRL-08768	SparkFun Electronics
2	Break Away Headers – Straight	PRT-00116	SparkFun Electronics
2	Break Away Female Headers	PRT-00115	SparkFun Electronics
2	Gas Sensor Breakout Board	BOB-08891	SparkFun Electronics

2	2mm 10pin XBee Socket	PRT-08272	SparkFun Electronics
1	Breakout Board for XBee Module	BOB-08276	SparkFun Electronics
1	iDigi™ X4 Starter Kit ZB	XK-Z11-IX	Digi International
2	LED, Red	COM-09590	SparkFun Electronics
3	LED, Yellow	COM-09594	SparkFun Electronics
2	LED, Green	COM-09592	SparkFun Electronics

Find Allied Electronics at: www.alliedelec.com.

Find SparkFun Electronics at www.sparkfun.com.

Find Digi International at: www.digi.com

Reminder: Download a ZIP file of source code, schematic diagram, and MATLAB files from: www.gfreak.com/GF167/GF167.zip.

-----END-----