# uGuru datasheet

First of all, what I know about uGuru is no fact based on any help, hints or datasheet from Abit. The data I have got on uGuru have I assembled through my weak knowledge in "backwards engineering".
And just for the record, you may have noticed uGuru isn't a chip developed by Abit, as they claim it to be. It's realy just an microprocessor (uC) created by Winbond (W83L950D). And no, reading the manual for this specific uC or mailing Windbond for help won't give any usefull data about uGuru, as it is the program inside the uC that is responding to calls.

Olle Sandberg <ollebull@gmail.com>, 2005-05-25

Original version by Olle Sandberg who did the heavy lifting of the initial reverse engineering. This version has been almost fully rewritten for clarity and extended with write support and info on more databanks, the write support is once again reverse engineered by Olle the additional databanks have been reverse engineered by me. I would like to express my thanks to Olle, this document and the Linux driver could not have been written without his efforts.

Note: because of the lack of specs only the sensors part of the uGuru is described here and not the CPU / RAM / etc voltage & frequency control.

Hans de Goede <j.w.r.degoede@hhs.nl>, 28-01-2006

## Detection

As far as known the uGuru is always placed at and using the (ISA) I/O-ports 0xE0 and 0xE4, so we don't have to scan any port-range, just check what the two ports are holding for detection. We will refer to 0xE0 as CMD (command-port) and 0xE4 as DATA because Abit refers to them with these names.

If DATA holds 0x00 or 0x08 and CMD holds 0x00 or 0xAC an uGuru could be present. We have to check for two different values at data-port, because after a reboot uGuru will hold 0x00 here, but if the driver is removed and later on attached again data-port will hold 0x08, more about this later.

After wider testing of the Linux kernel driver some variants of the uGuru have turned up which will hold 0x00 instead of 0xAC at the CMD port, thus we also have to test CMD for two different values. On these uGuru's DATA will initally hold 0x09 and will only hold 0x08 after reading CMD first, so CMD must be read first!

To be really sure an uGuru is present a test read of one or more register sets should be done.

## Reading / Writing

### Addressing

The uGuru has a number of different addressing levels. The first addressing level we will call banks. A bank holds data for one or more sensors. The data in a bank for a sensor is one or more bytes large.

The number of bytes is fixed for a given bank, you should always read or write that many bytes, reading / writing more will fail, the results when writing less then the number of bytes for a given bank are undetermined.

See below for all known bank addresses, numbers of sensors in that bank, number of bytes data per sensor and contents/meaning of those bytes.

Although both this document and the kernel driver have kept the sensor terminoligy for the addressing within a bank this is not 100% correct, in bank 0x24 for example the addressing within the bank selects a PWM output not a sensor.

Notice that some banks have both a read and a write address this is how the uGuru determines if a read from or a write to the bank is taking place, thus when reading you should always use the read address and when writing the write address. The write address is always one (1) more than the read address.

## uGuru ready

Before you can read from or write to the uGuru you must first put the uGuru in "ready" mode.

To put the uGuru in ready mode first write 0x00 to DATA and then wait for DATA to hold 0x09, DATA should read 0x09 within 250 read cycles.

Next CMD _must_ be read and should hold 0xAC, usually CMD will hold 0xAC the first read but sometimes it takes a while before CMD holds 0xAC and thus it has to be read a number of times (max 50).

After reading CMD, DATA should hold 0x08 which means that the uGuru is ready for input. As above DATA will usually hold 0x08 the first read but not always. This step can be skipped, but it is undetermined what happens if the uGuru has not yet reported 0x08 at DATA and you proceed with writing a bank address.

## Sending bank and sensor addresses to the uGuru

First the uGuru must be in "ready" mode as described above, DATA should hold 0x08 indicating that the uGuru wants input, in this case the bank address.

Next write the bank address to DATA. After the bank address has been written wait for to DATA to hold 0x08 again indicating that it wants / is ready for more input (max 250 reads).

Once DATA holds 0x08 again write the sensor address to CMD.

## Reading

---

First send the bank and sensor addresses as described above.
Then for each byte of data you want to read wait for DATA to hold 0x01 which indicates that the uGuru is ready to be read (max 250 reads) and once DATA holds 0x01 read the byte from CMD.

Once all bytes have been read data will hold 0x09, but there is no reason to test for this. Notice that the number of bytes is bank address dependent see above and below.

After completing a successful read it is advised to put the uGuru back in ready mode, so that it is ready for the next read / write cycle. This way if your program / driver is unloaded and later loaded again the detection algorithm described above will still work.

## Writing

---

First send the bank and sensor addresses as described above.
Then for each byte of data you want to write wait for DATA to hold 0x00 which indicates that the uGuru is ready to be written (max 250 reads) and once DATA holds 0x00 write the byte to CMD.

Once all bytes have been written wait for DATA to hold 0x01 (max 250 reads) don't ask why this is the way it is.

Once DATA holds 0x01 read CMD it should hold 0xAC now.

After completing a successful write it is advised to put the uGuru back in ready mode, so that it is ready for the next read / write cycle. This way if your program / driver is unloaded and later loaded again the detection algorithm described above will still work.

## Gotchas

---

After wider testing of the Linux kernel driver some variants of the uGuru have turned up which do not hold 0x08 at DATA within 250 reads after writing the bank address. With these versions this happens quite frequent, using larger timeouts doesn't help, they just go offline for a second or 2, doing some internal callibration or whatever. Your code should be prepared to handle this and in case of no response in this specific case just goto sleep for a while and then retry.

# Address Map

==========

## Bank 0x20 Alarms (R)

---

This bank contains 0 sensors, iow the sensor address is ignored (but must be written) just use 0. Bank 0x20 contains 3 bytes:

Byte 0:
This byte holds the alarm flags for sensor 0-7 of Sensor Bank1, with bit 0 corresponding to sensor 0, 1 to 1, etc.

Byte 1:
This byte holds the alarm flags for sensor 8-15 of Sensor Bank1, with bit 0 corresponding to sensor 8, 1 to 9, etc.

Byte 2:
This byte holds the alarm flags for sensor 0-5 of Sensor Bank2, with bit 0 corresponding to sensor 0, 1 to 1, etc.

## Bank 0x21 Sensor Bank1 Values / Readings (R)

This bank contains 16 sensors, for each sensor it contains 1 byte.
So far the following sensors are known to be available on all motherboards:
Sensor  0 CPU temp
Sensor  1 SYS temp
Sensor  3 CPU core volt
Sensor  4 DDR volt
Sensor 10 DDR Vtt volt
Sensor 15 PWM temp

Byte 0:
This byte holds the reading from the sensor. Sensors in Bank1 can be both volt and temp sensors, this is motherboard specific. The uGuru however does seem to know (be programmed with) what kindoff sensor is attached see Sensor Bank1 Settings description.

Volt sensors use a linear scale, a reading 0 corresponds with 0 volt and a reading of 255 with 3494 mV. The sensors for higher voltages however are connected through a division circuit. The currently known division circuits in use result in ranges of: 0-4361mV, 0-6248mV or 0-14510mV. 3.3 volt sources use the 0-4361mV range, 5 volt the 0-6248mV and 12 volt the 0-14510mV .

Temp sensors also use a linear scale, a reading of 0 corresponds with 0 degree Celsius and a reading of 255 with a reading of 255 degrees Celsius.

## Bank 0x22 Sensor Bank1 Settings (R)
## Bank 0x23 Sensor Bank1 Settings (W)

This bank contains 16 sensors, for each sensor it contains 3 bytes. Each set of 3 bytes contains the settings for the sensor with the same sensor address in Bank 0x21 .

Byte 0:
Alarm behaviour for the selected sensor. A 1 enables the described behaviour.
Bit 0: Give an alarm if measured temp is over the warning threshold (RW) *
Bit 1: Give an alarm if measured volt is over the max threshold (RW) **
Bit 2: Give an alarm if measured volt is under the min threshold (RW) **
Bit 3: Beep if alarm (RW)
Bit 4: 1 if alarm cause measured temp is over the warning threshold (R)

Bit 5: 1 if alarm cause measured volt is over the max threshold (R)
Bit 6: 1 if alarm cause measured volt is under the min threshold (R)
Bit 7: Volt sensor: Shutdown if alarm persist for more than 4 seconds (RW)
Temp sensor: Shutdown if temp is over the shutdown threshold (RW)

* This bit is only honored/used by the uGuru if a temp sensor is connected
** This bit is only honored/used by the uGuru if a volt sensor is connected
Note with some trickery this can be used to find out what kinda sensor is detected see the Linux kernel driver for an example with many comments on how todo this.

Byte 1:
Temp sensor: warning threshold (scale as bank 0x21)
Volt sensor: min threshold (scale as bank 0x21)

Byte 2:
Temp sensor: shutdown threshold (scale as bank 0x21)
Volt sensor: max threshold (scale as bank 0x21)

Bank 0x24 PWM outputs for FAN's (R)
Bank 0x25 PWM outputs for FAN's (W)
---

This bank contains 3 "sensors", for each sensor it contains 5 bytes.
Sensor 0 usually controls the CPU fan
Sensor 1 usually controls the NB (or chipset for single chip) fan
Sensor 2 usually controls the System fan

Byte 0:
Flag 0x80 to enable control, Fan runs at 100% when disabled.
low nibble (temp)sensor address at bank 0x21 used for control.

Byte 1:
0-255 = 0-12v (linear), specify voltage at which fan will rotate when under low threshold temp (specified in byte 3)

Byte 2:
0-255 = 0-12v (linear), specify voltage at which fan will rotate when above high threshold temp (specified in byte 4)

Byte 3:
Low threshold temp (scale as bank 0x21)

byte 4:
High threshold temp (scale as bank 0x21)

Bank 0x26 Sensors Bank2 Values / Readings (R)
---

This bank contains 6 sensors (AFAIK), for each sensor it contains 1 byte.
So far the following sensors are known to be available on all motherboards:
Sensor 0: CPU fan speed
Sensor 1: NB (or chipset for single chip) fan speed
Sensor 2: SYS fan speed

Byte 0:
This byte holds the reading from the sensor. 0-255 = 0-15300 (linear)

Bank 0x27 Sensors Bank2 Settings (R)
Bank 0x28 Sensors Bank2 Settings (W)
---

This bank contains 6 sensors (AFAIK), for each sensor it contains 2 bytes.

Byte 0:
Alarm behaviour for the selected sensor. A 1 enables the described behaviour.
Bit 0: Give an alarm if measured rpm is under the min threshold (RW)
Bit 3: Beep if alarm (RW)
Bit 7: Shutdown if alarm persist for more than 4 seconds (RW)

Byte 1:
min threshold (scale as bank 0x26)

Warning for the adventerous
==========

A word of caution to those who want to experiment and see if they can figure the voltage / clock programming out, I tried reading and only reading banks 0-0x30 with the reading code used for the sensor banks (0x20-0x28) and this resulted in a _permanent_ reprogramming of the voltages, luckily I had the sensors part configured so that it would shutdown my system on any out of spec voltages which proprably safed my computer (after a reboot I managed to immediatly enter the bios and reload the defaults). This probably means that the read/write cycle for the non sensor part is different from the sensor part.