UberGROM Interface

The UberGROM Interface is a GROM-compatible device built on the Atmel Mega-AVR series of microcontroller. It is self-contained and can attach to the TI-99/4A GROM bus with no additional hardware (although filter capacitors are recommended on the power supply lines). Although the code can be adapted to any size of AVR, the current build is shipped with the Atmel ATMEGA128 for its large flash memory size and rich peripheral interface.

The TI-99/4A has three types of memory built-in: CPU Memory, which is directly addressable by the CPU, VDP memory, only addressable through the VDP, and GROM memory, only addressable through the GROM memory base. Most homebrew cartridges are created using CPU memory, due to the difficulty in reproducing TI's proprietary GROM interface, and the inaccessibility of VDP memory from outside the system. This chip is intended to resolve this difficulty and allow GROM to be added easily to any cartridge.

The TI system provides direct connection for 64k of GROM space, divided into blocks of 8k each (most official GROMs used only 6k of this 8k space). The first three blocks are allocated to the operating system (GROM 0) and TI BASIC (GROM 1 and 2). The remaining 48k of space is intended for use by cartridges.

The system also provides the concept of paging to the GROM space. Rather than an explicit paging action being required, the CPU simply interfaces through a different set of addresses (a different GROM 'base'). The console scans 16 of these bases when building the selection list, and the memory map includes space for up to 256 bases. At 64k each, this represents a potential memory space of 16MB!

There are a few caveats to this, however. The three console GROMs are wired so that they always respond to any base, thus, only 48k is available in each base. In addition, there's no single microcontroller yet available that can provide this much memory, so multiple chips and additional circuitry is required to go to this entire range.

A second caveat is speed. While a ROM access to the cartridge base happens in 4 cycles (and that slow only due to the multiplexer), a GROM access can delay for as much as 30 times longer. This delay is unrelated to the performance of the microcontroller, as all console GROMs add their time to the delay, even if they are not required to respond to the access.

This microcontroller attempts to provide a medium between flexibility of these spaces, memory usage, and peripheral interface. It provides re-writable GROM memory, GRAM, EEPROM, GPIO, Analog-to-Digital converters, and a TTL serial port (USART).

SPECIFICATIONS:

Core: Atmel AVR ATMEGA1284 8-bit RISC @ 8MHz (internal clock)
GROM: 120k (in 8k banks, supports 16 access bases)
RAM: 15k (8k and 7k bank)

EEPROM: 4k
GPIO: 4 pins
ADC: 4
UART: 1
EEPROM Write Time: 3.4ms
EEPROM Erase Endurance: 100,000 cycles
Flash Erase Time: 72ms (entire 8k block)
Flash Write Time: 4.5ms (256 byte page)
Flash Erase Endurance: 10,000 cycles
Clock accuracy: +/- 2.5% @ 25C
Current Draw: approx. 8.5mA @ 25C

See the ATMega1284P Datasheet for more specifics.

The system can be used as a simple GROM device, or it can be used as a peripheral interface to external devices. Full source is available so that you can adapt it to your own usage. It's limited only by your own design!

This device is designed to work on the concept of memory-mapping. For each of the available GROM slots in the TI system, a configuration register specifies whether memory or peripheral is accessed at that address. All access to the device is done via GROM.

Terminology

GROM – Graphics Read Only Memory – TI's proprietary parallel, auto-incrementing memory device. Alternately, the memory range that would contain a single GROM on the TI. This device emulates many GROMs in one unit.

Base – The group of addresses that selects a bank of GROMs. Base 0 is at >9800 for 'read data', with offsets of 2 for 'read address', 1024 for 'write data', and 1026 for 'write address'. Subsequent bases are offset by 4 from the first, so, Base 1 is at >9804, Base 2 is at >9808, etc. This device supports 16 bases.

Slot – A term specific to this device to indicate the programmable memory address into which any peripheral may be configured.

The TI-99/4A GROM Interface

GROM is a parallel, auto-incrementing memory device. No matter how many GROMs are physically attached to a base, all GROMs see all GROM accesses and all of them respond in some manner. This section covers only the software interface. For information on the hardware interface, it is recommended to consult additional sources, such as Thierry Nouspikel's excellent reference at http://www.nouspikel.com/ti99/titechpages.htm

Four addresses are dedicated in the 9900's address space for communicating with GROM. This is called the GROM 'base'.

>9800 – read data
>9802 – read address
>9C00 – write data
>9C02 – write address

Normal operation of the GROM starts by writing an address. This is done by sending the two bytes of the desired address to the write address base (with simple MOVB opcodes), sending the least significant byte first. Once the address has been sent, the data can be read by reading from the Read Data base (again, with MOVB). The GROM automatically increments its internal address every time a byte is read, so that subsequent bytes can be read without having to set the address again. When reading, only the GROM that actually contains the data will present it, but all GROMs participate in the cycle.

The auto-increment of the GROM address has one caveat, when it comes to thinking of the address space. Although there are 16 bits of address space, each GROM tracks only 13 bits of address internally (for 8k of space). The other 3 bits are latched as a 'device select'. This means that when an address reaches that 8k boundary, it wraps around to the beginning of that GROM again, rather than going into the next boundary. This is a bit counter intuitive, but it makes sense when you remember that is it normally a bank of 8k (or 6k!) chips all responding to the same bus. This internal address management allows the chips to be used without any external hardware for chip selection, but makes it difficult to automatically count from one chip to the next. The only way to change which 8k block is active is to write a new GROM address. It is important to be aware of this behavior with the peripherals, which may wrap around back to GROM addresses if your code does not follow the described rules.

There is an advanced configuration option to disable this behavior and allow the address to increment within the space of the device (>6000->FFFF).

GROMs are notoriously slow devices, but unlike the VDP, the software need not be aware of delays when accessing them, as they will block the CPU for as long as is needed to complete the operation. Although this microcontroller is much faster than the real GROMs for most operations (flash writing excluded), the delay still extends to the slowest GROM in the system.

Read Address is used to read the current address in the GROMs back, and again, the LSB is sent first. Reading the address corrupts it, and so a new address must be written before you may read again. Due to the GROM internally prefetching data, the retrieved address will be one higher than expected. Note that reading address on peripheral blocks may not return the expected address. This should only be used on banks emulating GROMs or GRAMs (or real GROMs, of course).

Note that this device does not return addresses, therefore, there must always be at least one real GROM in the system.

Write Data is not used on normal GROMs, but may be used with GRAM and with this microcontroller in the same manner. The effect of prefetch on the address may be safely ignored when writing, but as the standard operation for this was never defined by TI, it is not recommended to mix reads and writes without resetting the address in-between.

To access a different GROM base, just add 4 to the desired address for each base. For instance, the above addresses access base 0, base 1 would use >9804 for Read data, base 2 would use >9808, and so forth. Remember that the console GROMs (>0000 - >5FFF) respond regardless of the base number.

A note on device interfacing

The TI console includes limited support for multiple GROM bases, and a 'Review Module Library' option. However, it determines whether to activate this by comparing the first 31 bytes of cartridge space, starting at >6000. There is a bug in this loop, in that if multiple bases ARE detected, but no valid cartridge header is detected, the console will go into an infinite loop while building the module list. In order to make it less likely to cause this behavior by accident, all peripheral spaces on this device start at a >0020 offset, and the first 32 bytes. Thus, only GROM, RAM or EEPROM can actually be at >6000.

If you want to avoid the console detecting multiple GROM bases, and thus prevent "REVIEW MODULE LIBRARY" from appearing on the cartridge menu, simply select the same memory at GROM address >6000 in both bases 9800 and 9804 – this is all that the console checks. This costs you one slot, but there are plenty of spare slots even if you wish to map all peripherals.

EEPROM/Configuration space

EEPROM stands for 'electrically erasable programmable read-only memory'. The EEPROM in this device is a non-volatile memory that can be read and written a single byte at a time, using standard GROM accesses, although writes may be notably slower. The first few bytes of the EEPROM contain paging and configuration registers, the rest can be used by your own application. This data is non-volatile and does not require power or a battery to retain. (Although, as EEPROM data can fail over repeated use, it is recommended that a checksum or other system always be used to validate the stored data.)

It should be noted that the EEPROM is rated for a limited number of writes to a particular byte, specifically 100,000. This is adequate for tasks such as saving configuration data, high scores for games, , occasional changing of the cartridge by a user, etc, but not for continuous access tasks, such as bank switching during execution. As such, the configuration block here is intended to be set once by the cartridge creator, or by a user loading a new cartridge into their device, but not continuously during runtime. (Bank-switching with a cost of 3.4ms per switch would be unreasonably slow anyway). Rather than switching devices in and out, simply use the multiple GROM base feature to map all devices into unique addresses, and they will all be available to your application simultaneously.

The configuration EEPROM is available in top 2k of the last GROM base (even if GROM bases are not being used), starting at GROM address >F800, regardless of configuration. (Base 15 is accessed starting with CPU address >983C). This mapping is fixed and cannot be moved, as it is the only way to access the banking configuration. This means that GROM space in base 15 is limited to 38K. (However, if banking is not used, the entire 40k space of bank 0 is still available). As the EEPROM is larger than 2k (see specifications), then it must be mapped into another base to access the rest of the data.

The configuration EEPROM defines the following registers, which start at >F800 in base 15 in GROM space (always), or at >0000 in the GROM base that EEPROM is mapped to.

0000 – Configuration flags. The following bits are defined. Unused bits should be set to 0

01 – Enable GROM bases when set. When disabled, the GROM will respond to all bases, the same way that the console GROMs do. In this mode, only 16 bytes are used for mapping. When set, the GROM will respond to the first 16 GROM bases independently, and 256 bytes are used for mapping. Note that regardless of this setting, the fixed EEPROM space is available on base 15 at >F800

02 – Disable Recovery program when set. On first powerup, the chip will use the powerup vector at >E000 to test the keyboard, and if space is being held, it will load a simple test GROM at >6000 instead of whatever is configured. This behavior is probably not desired for a production cartridge, but should not affect any other. Set this bit to prevent it. Note that if you set this bit, you will need an external way (such as a ROM or boot menu) to load the loader program to change anything on the cartridge, as the internal loader will not be accessible.

04 – Allow rollover. TI manufactured GROMs maintain an address counter within an 8k grom. That is, when they reach the end of their slot, they wrap back to the beginning. (So, for instance, >7FFE, >7FFF, >6000, >6001, etc). Many GROM emulation devices did not have this behavior, they maintained a single 16-bit address counter (thus allowing 0x7FFE, 0x7FFF, 0x8000, 0x8001, etc). By default, this device emulates the original behavior. If you set this bit, then roll-over into the next slot will be permitted. In order to prevent conflict with the console GROMs, roll-over from 0x5FFF->0x6000, and from 0xFFFF->0x0000 will not occur in either case. Note: Because this device does not provide address read-back (this functionality is provided still by the console GROMs), if you roll over from one GROM slot to the next the system GROM address will be incorrect if read back. Therefore, ensure that this will not be a problem for your application before using this bit. For instance, it may be a problem for GPL applications which read back frequently, but less so for data copy loops and the like. (Example: If you allow the GROM read to increment from >7FFF to >8000, data will continue to come from the correct place, but reading the GROM address will return >6000 instead of >8000, because the console GROMs do not support this operation.)

0001 – Inverse bits to register 0000. If this value is not the same as a bitwise invert of register 0000, then the value in 0000 is considered invalid and not used. (0x00 is assumed).

0002 – Map configuration for GROM slot >0000. Note: this slot is reserved for the TI operating system, and any override placed here, in any base, is ignored. (For future expansion)

0003 – Map configuration for GROM slot >2000. Note: This slot is reserved for the console TI BASIC, and any override here, in any base, is ignored. (For future expansion)

0004 – Map configuration for GROM slot >4000. Note: This slot is reserved for the console TI BASIC, and any override here, in any base, is ignored. (For future expansion)

0005 – Map configuration for GROM slot >6000.
0006 – Map configuration for GROM slot >8000.
0007 – Map configuration for GROM slot >A000.
0008 – Map configuration for GROM slot >C000.
0009 – Map configuration for GROM slot >E000.

000A – Inverse bits to register 0002. If this value is not the same as a bitwise invert of register 0001, then the value in 0001 is considered invalid and not used.
000B – Inverse bits to register 0003.
000C – inverse bits to register 0004.
000D – inverse bits to register 0005.
000E – inverse bits to register 0006.
000F – inverse bits to register 0007.
0010 – inverse bits to register 0008.
0011 – inverse bits to register 0009.

0012-0101 – Map configuration for GROM slots >0000->E000 in cases where multiple GROM bases are enabled. 16 bytes are used for each slot in the same manner as defined above for the first base. Note that when multiple GROM bases are enabled, the TI console GROM requires at least ONE valid cartridge GROM to be available, otherwise the title selection menu will hang. If multiple GROM bases are not enabled, this memory is free for user data.

0102-max – Free for user data. Note that the hardware write-protect pin will protect the configuration space from 0000-0101, but leave this user space available for updating. (Added 6/2/2015)

The following table lists the position in the table for each combination of base (down) and slot (across) to load in the EEPROM. All address offsets are in hexadecimal, and the inverted byte offset is gained by adding ‘8’ to the listed value.

	0	2000	4000	6000	8000	A000	C000	E000
0	2	3	4	5	6	7	8	9
1	12	13	14	15	16	17	18	19
2	22	23	24	25	26	27	28	29
3	32	33	34	35	36	37	38	39
4	42	43	44	45	46	47	48	49
5	52	53	54	55	56	57	58	59
6	62	63	64	65	66	67	68	69
7	72	73	74	75	76	77	78	79
8	82	83	84	85	86	87	88	89
9	92	93	94	95	96	97	98	99
10	A2	A3	A4	A5	A6	A7	A8	A9
11	B2	B3	B4	B5	B6	B7	B8	B9
12	C2	C3	C4	C5	C6	C7	C8	C9
13	D2	D3	D4	D5	D6	D7	D8	D9
14	E2	E3	E4	E5	E6	E7	E8	E9
15	F2	F3	F4	F5	F6	F7	F8	F9

Map Configuration

The map configuration consists of two nibbles. The first nibble indicates the device type to be mapped in, devices are described below. The second nibble indicates which page of the device, if it has multiple pages. For instance, a map of >12 indicates GROM page 2 should be mapped. The number of pages available is the device memory size divided by 8k, rounded up.

Device ID	Type	Access	Pagable?
0	RAM	Read/Write	Yes
1	GROM	Read Only	Yes
2	EEPROM	Read/Write	No
3	GPIO	Peripheral	No
4	ADC	Peripheral	Yes
5	UART	Peripheral	No
6	Flash	Peripheral	No
7	Timer	Peripheral	No

Non-pagable devices should set a page of 0 for compatibility.

Device Type 0 – RAM

This microcontroller provides a block of ordinary RAM memory which may be read or written through the standard GROM interface (as if GRAM). No locking or erasing is required, and the data is available upon paging. The data is volatile, however, and will be lost if the console is turned off or the cartridge is removed from it.

It is even possible to load GROM programs to the RAM block and have the interpreter execute them, if desired. However, because this RAM is on the GROM bus, the TI CPU cannot execute 9900 assembly code from it.

Each RAM page is 8k in size – in this controller the second page is only 7k in size.

Device Type 1 – GROM

This is the standard GROM data, and will generally consist of many available pages of read-only memory. This memory is stored in the microcontroller's flash and is non-volatile, it does not require power to retain. It is not trivially re-writable, you must page in the Flash device and explicitly erase and re-write blocks of data. In this mode (type 1), however, it will behave like standard GROM. Writes to the GROM data will be ignored.

Note: There is one limitation on the placement of data in GROM. A Powerup link placed at >E000 in the first GROM base will not execute unless the recovery code is disabled in the configuration space. See the Recovery code section.

Device Type 2 – EEPROM

See the section above about EEPROM/Configuration data for full details. This page can be mapped into a block for convenience or to access more than 2k of EEPROM, if available. Otherwise, it is always available at GROM address >F800 in bases 15 (even if GROM bases are not enabled in configuration).

The EEPROM space will be read-only until an unlock sequence is written to it. To unlock it, simply write to >FFFF on base 15 three times, with the following bytes. Note that any other GROM write access to EEPROM space will break this sequence (but by limiting it to EEPROM space it should still work from GPL): Write 0x55, then write 0xAA, then write 0x5A.

You can lock the EEPROM again by writing any other value to >FFFF on base 15. This is recommended after a modification to prevent resets and glitches from accidentally modifying EEPROM data.

Be careful if accessing user EEPROM in the default mapping on base 15, since only offsets 0x0000-0x07FE are available – offset 0x07FF overlaps the lock control address. This does not apply when EEPROM is mapped anywhere else.

Device Type 3 – GPIO

GPIO stands for General Purpose Input/Output. This microcontroller supports 4 GPIO pins. Each pin is a 5-volt pin which may be configured for input or output. When input, it may also be configured to activate an internal pull-up resistor, to reduce external parts count.

The device registers start at >0020 in whichever page they are mapped into. In addition, the access space deliberately repeats over a range to allow repeated access to the I/O pins without needing to change the GROM address between each one. (Even though this is a device access, the GROM address must still auto-increment).

The least significant bit indicates GPIO0 in all cases.

0020 – Configuration bits for Input or Output. Each bit, starting with the LSB, is set to 0 for input, or 1 for output. On reset, all GPIO are set to input with no pull-up – your software must configure it.

0021-1FFF – GPIO pin access. When reading, returns the current input state of each pin, with the LSB set to GPIO0. This is true whether the pin is in input or output mode. When writing a pin set for output, the pin will output 5v if a 1 bit is set, or 0v if a 0 bit is set. When writing a pin set for input, the pull-up resistor is enabled if a 1 bit is set, or disabled if a 0 bit is set.

The large range is used to allow multiple accesses without resetting the GROM address, but note that to read or write continuously, you must reset the GROM address before it leaves this range.

Device Type 4 – ADC

ADC stands for Analog to Digital Conversion. The microcontroller supports multiple analog input pins with a sense range from 0 to 5v, with 8-bit resolution (depending on the system configuration.) The ADC 'page' mapped indicates which of the 4 ADCs will be accessed.

The ADCs are mapped starting at an offset of >0020. The entire memory space (>0020 - >1FFF) repeats to allow them to be read continuously without needing to reset the GROM address. The page number specified indicates the ADC that you wish to access.

Reading the ADC address triggers the conversion immediately, which will convert in approximately 104uS. A value of 0 indicates a value near 0v, while 255 indicates a value near 5v.

Note: the first conversion after power up will take approximately 200uS.

Tip: By placing the ADCs in different GROM base bases, but at the same GROM address, you could read all ADCs without having to update the GROM address just by changing which CPU address you access GROM through.

Device Type 5 – UART

UART stands for Universal Asynchronous Receiver/Transmitter, and this is a serial communication device. The UART in this microcontroller can handle high speed bidirectional traffic, and is equipped with two 255 byte buffers, one each for receive and transmit. Despite this, higher speeds may have limited value on the TI due to its ability to actually service the buffers without losing data.

The UART is mapped starting at an offset of >0020. Also, the read and write bases are allocated a block of repeating memory to allow them to be accessed continuously without needing to reset the GROM address.

0020 – Status/Configuration flags:

> 01 – (read only) When set, there is data available to be read. If the buffer fills and more characters arrive, the additional characters are dropped (and set the Data Overrun bit).
>
> 02 – (read only) when set, there is room in the output buffer for at least one byte of data.

04 – (Read Only) Tx Buffer Empty – Indicates that the transmit buffer is completely empty (ie: all data is sent)

10 – (read only) Frame error - This bit is set if the USART had a Frame Error. I.E. when the first stop bit of the next character in the receive buffer is zero. There is no way to tell which character in the buffer this applied to, and due to the behavior of the hardware, in some cases it may not clear until a valid character is received. This bit may be of limited value.

20 – (read only) Data Overrun – this bit is set if the receive buffer is full and a new character is received. At very high bit rates characters may be lost without moving to the buffer, in which case this bit is not set, but this case should be very rare.

40 – (Read only) Parity Error – this bit is set if a received character had a parity error when it was received. For the same reasons as Frame Error, this bit may be of limited use.

0021 – Line setting flags:

01-02 – Number of bits. 11=8,10=7,01=6,00=5

04-08 – Parity generation. 00=none,01=none,10=even,11=odd

10 – Stop bits. 0=1 stop bit, 1=2 stop bits

20 – 2X mode (doubles the baud rate at the expense of less accurate clock tolerance).

0022-0023 – Speed setting. This is a Little-Endian, 12-bit value loaded into the AVR's UBRR register to set the baud rate of the serial device. To calculate this value, use: UBBR = (8,000,000/(16*BAUD))-1. The valid range is from 0 (representing about 500,000bps) to 4095 (representing about 122bps). Rates are doubled (but clock recovery accuracy halved) if the 2X bit is enabled. You must write these bytes in order – LSB then MSB. The change won't take effect until the MSB is written.

0024 – number of bytes in the RX buffer. This can be used to optimize reads from the buffer – even if new bytes are added, this much data is guaranteed to be present.

0025 – number of bytes in the TX buffer. This can be used to optimize writes to the buffer – this much space is guaranteed to be present.

0100-0FFF – Write byte (goes to buffer before transmission). If the TX buffer is full, the byte is silently dropped – you should check the buffer before writing. Note this GROM range is smaller than for reading.

1000-1FFF – Read byte (from buffer if buffering is on). Note that this will clear error bits in the status/configuration register. If no character is available in the buffer, the returned byte is not guaranteed – you should check the buffer before reading.

Device Type 6 – Flash Controller

The flash controller provides write access to the GROM memory space. The grom slot is the same as for the GROM device type. In this mode, you can erase the page and rewrite data 256 bytes at a time. This mode can be disabled with the external Write Protect hardware pin.

The flash controller is mapped starting at an offset of >0000.

The process is fairly simple:

1) Check the result code for write protect (abort if set)
2) Set the page select register and slot select register
3) Write the erase command to the command register (will delay until finished)
4) Check the result code for errors
5) Write the new page data to the write buffer
6) Write the program command to the command register (will delay until finished)
7) Check the result code for errors
8) Repeat until all pages are written

0000-00ff – Write buffer (one page worth – write only). You MUST write this page sequentially, and always write an even number of bytes, for correct results. This buffer is write-only.

0100 – Page select in GROM slot (0-31)

0101-0102 – Write command register. Write the command in 0201 and the inverse in 0202 (write only):

31/CE = Erase block – causes the page (256 bytes) to be erased to 0xff
D2/2D = Write buffer – writes the contents of the write buffer to the current page

0103 – Result Code (read-only)

0 = Idle
1 = Busy
2 = Write Protected
3 = Command Error

0x0104 – GROM slot (0-15)

Device Type 7 – Timer

The timer is a free-running, real-time tick counter that can be used to time events independent of the console operating speed, and with a higher resolution than the VDP interrupt (and with a quicker readback than the 9901 timer). The timer runs at 7812.5Hz (+/- 2.5%), and is 16-bits in size.

The timer is mapped starting at an offset of >0020. Also, the read-back registers are allocated a block of repeating memory to allow them to be accessed continuously without needing to reset the GROM address.

Because the timer is 16-bits in size, two reads are required to read the data. Every even numbered address is the LSB, and every odd numbered address is the MSB. The timer values are updated when the LSB is read, thus, the MSB will always be associated with the LSB. Because of this, you cannot simply read the MSB, the value will never change.

0020-1FFF – Timer value, 16-bit little-endian (LSB,MSB), repeating (read only).

Recovery Program

The device includes a small recovery program to facilitate re-loading the chip even if suitable software is not otherwise installed. A PowerUp link is always inserted on the FIRST powerup only, at the >E000 GROM base. (That is, it will not be re-executed when FCTN-= or even a TI Hardware Reset is executed, the system must be power cycled to re-trigger it). If the user holds space during the power-on, the GROM at >6000 on base 0 is replaced with the recovery program, which contains a copy of Easy Bug, and the LOAD PROGRAM IMAGE function from Editor/Assembler. This allows troubleshooting and the ability to load any piece of software into the system for the purpose of configuring the chip.

If this functionality is not desired, for instance, either you need a real power up routine in the GROM at >E000, or you are distributing a cartridge and do not want users to be able to activate this mode, simply set bit value >02 in the configuration bits. Once you do this, it will be impossible to reactivate this recovery program without updating this configuration bit again – since the GROM chip usually occupies the cartridge base, this means you may need an alternate way to load software (such as a ROM beside the GROM, or boot software like the startup MENU program.)

The recovery program contains program software copyright by Texas Instruments, and distributed under license. TI makes no warranty with respect to the programs and is under no obligation to provide any support or assistance with respect to the programs. TI is under no obligation to provide upgrades to the programs. No liability is accepted on the part of Texas Instruments or the author with respect to use, copying or distribution of the programs. This software has been modified from its original form.