AVGA

AVR based open source color video game development platform
for single chip game console
http://avga.prometheus4.com

v 0.2 API description
- work in progress -

PART 1. *THE CORE*

1.1 Video signal generator API

1.1.1 Initialization

void video_init (sync_t *s);

Description: Initializes the configured sync generator.
Parameters: ***s:** pointer to sync_t pulse array in PGM that describes the video sync pattern to be generated.
VGA, PAL and NTSC signals are defined in video.h for this purpose.
Note: Function can be called many times. This is useful for example for swictching between the standards in runtime.
When changing the video standard (or resolution), one should always check the following:
1. VIDEO_YPOS (config_screen.h) - this value must be lower or equal than active line count in the video standard.
2. The nominal line period must be greater than DRIVER_RESX*5 + overhead (in clocks). If this condition is not met, the scan line interrupt can overflow. Then DRIVER_WIDTH (config_screen.h) should be decreased.

Example:

```
int main(void)
{
        driver_mmap(table, pgmmap, rammap);
        video_init(PAL);
        enable();

        //place your game code here
}
```

unsigned int sync ( );

Description: Waits for retrace. This routine will return just after the last line was scanned. When finishes, it is the right time for immediate drawing a new frame to the screen, or flipping the reference table (double-buffering).
Returns: Amount of time spent within this function (for time measurement).

Example:

```
while(1)        //game loop
{
        render();
        frameMove();
        sync();
}
```

enable ( );

Description: Enables the engine (video output). Since no other interrupts are allowed, this does just "sei".

disable ( );

Description: Disables the engine (video output). Since no other interrupts are allowed, this does just "cli".

1.1.2 Screen effects.

Note: Theese macros/routines are valid only when NOT using WINDOW tool. Each window can be handled individually in the same way.

video_show ( );

Description: Shows the screen.

video_hide ( );

Description: Hides the screen.

void video_set_colormask (unsigned char mask);

Description: Sets the color mask for the screen. This will affect the Data Direction Register of the video port.

Parameters: **mask** – The desired color mask, lower 4 bits used only. The actual DDR value for lower 4 bits of the video port can be set with VIDEO_DEFAULT_DDR in config_hw.h. However, the driver toggles all bits of the port. So some scrambled pixels will appear on theese bits when set as outputs.

void video_scroll (unsigned char x, unsigned char y);

Description: Scrolls the screen x pixels horizontally and y pixels vertically.

Parameters: **x:** amount of pixels to scroll along horizontal axis

	y: amount of pixels to scroll along vertical axis
Note:	The maximum supported scrolling value is (DRIVER_BLOCK_HEIGHT-1) in Y direction and (DRIVER_BLOCK_WIDTH-1) in X direction.

void video_set_scroll (unsigned char x, unsigned char y);

Description:	Sets the scroll values for the screen.
Parameters:	**x:** scrolling along horizontal axis
	y: scrolling along vertical axis
Note:	The maximum supported scrolling value is (DRIVER_BLOCK_HEIGHT-1) in Y direction and (DRIVER_BLOCK_WIDTH-1) in X direction.

video_adjust_for_background (x, y);

Description:	Sets the scroll values for the screen with one's complement of x and y.
Note:	Useful for rendering a background image with smooth scrolling.
Parameters:	**x:** reversed scroll value along horizontal axis
	y: reversed scroll value along vertical axis

Example:

```
void RenderMap ( int x, int y )
{
        video_adjust_for_background (x, y);
        background_draw ( x>>3,  y>>3, map, WIDTH, HEIGHT );
}
```

video_set_startline (n);

Description:	Sets the start line number. I.e. the first line from the video memory that appears on your monitor.
Parameters:	**n:** the desired line number

video_move (n);

Description:	Moves the screen's content along vertical axis (increases/decreases the start line number).
Parameters:	**n:** the amount of lines (pixels) to move.

1.2 Video graphics driver API

1.2.1 Initialization

void driver_mmap (void *scrp, PGM_P pgmp, void *ramp);

Description: The function will map all the default memory areas for the graphics driver.

Parameters: **scrp:** Default reference table (video ram) pointer. The driver assumes that free RAM space is available at this pointer with size DRIVER_REFTABLE_SIZE. This is not a rule, when using window-individual reference tables. In such case, it can be even NULL. All newly created windows will have this reference table by default.
pgmp: Default PGM map (flash tileset) pointer. This should point to FLASH at tile graphics defintition. This is not a rule, when using window-individual reference tables. In such case, it can be even NULL. All newly created windows will have this tileset by default.
ramp: RAM map (ram tileset) pointer. It is supposed that free ram space is available on this pointer with size DRIVER_RAMMAP_SIZE.

Note: Funtion should be allways called before the first enable of the video generator. You can use following macros: driver_map_pgmp(pgmp), driver_map_scrp(scrp), and driver_map_ramp(ramp) for maping each of theese spaces individually.

Example:

```
//memory for the driver.
unsigned char screen[DRIVER_REFTABLE_SIZE];
const unsigned char pgmmap[] PROGMEM = TILESET;
unsigned char rammap[DRIVER_RAMMAP_SIZE];

int main(void)
{
	driver_mmap(table, pgmmap, rammap);
	video_init(PAL);
	enable();

	//place your game code here
}
```

1.2.2 I/O functions

Note: The following macros/routines works with the driver's default video memories. I.e. if a window has its individual reference table and/or tileset mapped elsewhere, it will not be affected. Each window can be handled individually in the same way.

driver_get_tileset ( );

Description:	returns back the default tileset (see driver_mmap(..))
Returns:	Pointer to the tileset.

driver_get_reftable ( );

Description:	returns back the default reference table (see driver_mmap(..))
Returns:	Pointer to the reference table (video ram).

driver_get_rammap ( );

Description:	returns back the ram map (see driver_mmap(..))
Returns:	Pointer to the ram map.

driver_rammap_block (x);

Description:	Maps the given RAM block to ram.
Parameters:	**x:** the number of block in rammap.
Returns:	ID of the block mapped to ram

Example:
```
//show tile number 5 from RAM at x, y.
driver_set_block ( x, y, driver_rammap_block ( 5 ) );
```

driver_is_rammap (i);

Description:	Check if the given block ID is in mapped to ram.
Parameters:	**i:** tile ID
Returns:	TRUE: block is mapped to RAM
	FALSE: block is mapped to PGM

driver_get_block_ptr (x, y);

Description:	Gets pointer from reftable to block at x, y
Parameters:	**x, y:** tile coordinates.
Returns:	Pointer to the block.

driver_get_block (x, y);

Description:	Gets ID of block at x, y.
Parameters:	**x, y:** tile coordinates.
Returns:	ID number.

driver_set_block (x, y, blk);

Description: Sets the tile at x, y to block BLK.
Parameters: **x, y:** tile coordinates.
blk: ID of the block.

driver_fill (blk);

Description: Fills the screen with block blk
Parameters: **blk:** ID of the block.

driver_clear ( );

Description: Clears the whole screen (Fills with EMPTY_BLOCK)
Note: The EMPTY_BLOCK id constant can be redefined.

driver_print (x, y, txt);

Description: Prints RAM string.
Parameters: **x, y:** left-top start tile coordinates.
txt: pointer to the null-terminated string in the RAM.

driver_print_P(x, y, txt);

Description: Prints PGM string.
Parameters: **x, y:** left-top start tile coordinates.
txt: pointer to the null-terminated string in the flash.

driver_print_C(x, y, string);

Description: Allocates the string constant to PGM and prints it.
Parameters: **x, y:** left-top start tile coordinates.
string: the string constant.

Example: **driver_print_C(0, 0, “Hello, world!“);**

driver_get_pgmblock_ptr (i);

Description: Gets pointer to block graphics in the tileset (pgmmap).
Parameters: **i:** tile ID (number of block in PGM)
Returns: RAM pointer to the block graphics.

driver_get_ramblock_ptr (i);

Description:	Gets pointer to block graphics in rammap.
Parameters:	**i:** number of block in RAM.
Returns:	PGM pointer to the block graphics.

1.3 Sound sequencer API

1.3.1 Initialization

void sound_init ( );

Description:	Initializes the configured sound sequencer.
Note:	The sequencer needs the video generator running.

1.3.2 Playback

void sound_play (PGM_P sndfile);

Description:	Plays the audio sequence stored in PGM.
Parameters:	**sndfile:** pointer to the defined sequence in PGM.
Note:	File format consists of 2 byte sentences. First byte specifies duration of the note (video frames) and the second defines wave period (1/frequency). When the period is zero, sound is muted (pause). Zero duration is considered as end of file mark.

Example:

```
const unsigned char sound1 [] PROGMEM =
{
        5, 200,
        5, 180,
        5, 160,
        5, 140,
        0
}

sound_play ( sound1 );
```

void sound_stop ( );

Description:	Stops the playback.

PART 2. THE UTILS

2.1 Window API

2.1.1 Creating/handling windows

WINDOW window_add(unsigned char line, unsigned char height, unsigned char params);

Description: Adds a general new window to the screen.
Parameters: **line:** number of the first (top) line in the reference table for this window
height: the height of window in pixels
params: parameters for the window
Returns: Handle to the created window
Note: All the widndows are aligned from top of the screen in order they were added.
Maximum number of windows on is defined in config_utils.h.
Following macros should be use instead of this function.

WINDOW window_add_visible (line, height);

Description: Adds a visible window to the screen.
Parameters: **line:** number of the first (top) line in the reference table for this window
height: the height of window in pixels
Returns: Handle to the created window.

WINDOW window_add_standard (row, rows);

Description: Adds a Standard (scrollable) visible window to the screen.
Parameters: **row:** the first row from the window's reference table to show
rows: height of window in rows
Returns: Handle to the created window.
Note: For scrolling, there must be at least one more row left int the window's reference table.
Example:

```
WINDOW wnd1 = window_add_standard(0, 15);
WINDOW wnd2 = window_add_standard(16, 3);  //leave one row for scrolling
```

WINDOW window_add_space (height);

Description: Adds an empty (invisible) window to the screen.
Parameters: **height:** the height of window in pixels
Returns: Handle to the created window.

window_reset ( );

Description: Clears all windows and restores original state (default reference table mapped to whole screen)

window_select (WINDOW wnd);

Description: Selects a window for the operations in the 2nd variant.
(see *Manipulating windows*)

2.1.2 *Manipulating windows (I/O, effects, scrolling)*

Note: All the following macros / routines are available in two variants:

1. Operations taking the window to work with as an argument (a – prefix).
2. Optimized operations worikng with the selected window.

WINDOW awindow_add_clone (wnd);
WINDOW window_add_clone ( );

Description: Creates a clone of the given window on the screen.
Returns: Handle to the created window.

awindow_swap_content (wnd1, wnd2);
window_swap_content (wnd2);

Description: Swaps the content of the two windows (reference table).
Parameters: **wnd2:** handle to the second window.
Notes: This is useful for example for double buffering.

window_get_reftable ( );
window_get_ptr ( );
awindow_get_reftable (wnd);
awindow_get_ptr (wnd);

Description: Returns back a reference table which this window is mapped to.
Returns: Pointer to the reference table.

window_set_reftable (ptr);
awindow_set_reftable (wnd, ptr);

Description:	Sets a new reference table for this window.
Parameters:	**ptr:** pointer to the reference table.
Note:	Valid only when WINDOW_INDIVIDUAL_REFTABLE option is enabled.

window_get_tileset ( );
awindow_get_tileset (wnd);

Description:	Returns back a tileset used by this window.
Returns:	Pointer to the tileset in FLASH.

window_set_tileset (ptr);
awindow_set_tileset (wnd, ptr);

Description:	Sets a new tileset for this window.
Parameters:	**ptr:** pointer to the tileset in FLASH.
Note:	Valid only when WINDOW_INDIVIDUAL_TILESET option is enabled.

window_get_row ( );
awindow_get_row (wnd);

Description:	Returns back first shown row of tiles in window’s reference table.
Returns:	number of the first row

window_get_startline ( );
awindow_get_startline (wnd);

Description:	Returns back first shown video line in window’s reference table.
Returns:	number of the video line

window_get_num_rows ( );
awindow_get_num_rows (wnd);

Description:	Returns back height of this window in TILE ROWS.
Returns:	number of the rows.

window_get_num_lines ( );
awindow_get_num_lines (wnd);

Description:	Returns back height of this window in lines (pixels).
Returns:	number of the pixels.

window_get_scrollX ( );
awindow_get_scrollX (wnd);

Description:	Returns back number of pixels scrolled along horizontal axis.
Returns:	number of the pixels.

window_get_scrollY ( );
awindow_get_scrollY (wnd);

Description:	Returns back number of pixels scrolled along vertical axis.
Returns:	number of the pixels.

window_get_block (x, y);
awindow_get_block (wnd, x, y);

Description:	Returns back block in this window at x, y.
Parameters:	**x, y:** tile coordinates.
Returns:	the block ID.

window_set_block (x, y, blk);
awindow_set_block (x, y, blk);

Description:	Sets the block in this window at x, y.
Parameters:	**x, y:** tile coordinates.
	blk: block ID to set

window_fill (blk);
awindow_fill (wnd, blk);

Description:	Fills the window with given block
Parameters:	**blk:** block ID to fill with

window_fill_scroll (blk);
awindow_fill_scroll (wnd, blk);

Description: Same as *window_fill(blk);* but fills one more row after the last one.
Parameters: **blk:** block ID to fill with.

window_print (x, y, txt);
awindow_print (wnd,x, y, txt);

Description: Prints null-terminated string of block IDs from RAM to the window.
Parameters: **x,y:** left-top start tile coords.
txt: pointer to the RAM string

window_print_P (x, y, txt);
awindow_print_P (wnd,x, y, txt);

Description: Prints null-terminated string of block IDs from PGM to the window.
Parameters: **x,y:** left-top start tile coords.
txt: pointer to the PGM string

window_print_C (x, y, txt);
awindow_print_C (wnd, x, y, txt);

Description: Allocates a null-terminated string of block IDs to PGM ands prints it to the window.
Parameters: **x,y:** left-top start tile coords.
txt: string constant
Example: window_print_C(0, 0, „Hello, world from the window!“);

window_show ( );
awindow_show (wnd);

Description: *Shows the window on the screen.*

window_hide ( );
awindow_hide (wnd);

Description: *Hides the window on the screen.*

window_set_colormask (mask);
awindow_set_colormask (wnd, mask);

Description:	Sets the color mask for the window. This will affect the Data Direction Register of the video port.
Parameters:	**mask:** The desired color mask, lower 4 bits used only. The actual DDR value for lower 4 bits of the video port can be set with VIDEO_DEFAULT_DDR in config_hw.h. However, the driver toggles all bits of the port. Undefined data stream will appear on theese bits when set as outputs.

window_set_scroll (x, y);
awindow_set_scroll (wnd, x, y);

Description:	Sets the scroll values for the window.
Parameters:	**x:** scrolling along horizontal axis
	y: scrolling along vertical axis
Note:	The maximum supported scrolling value is (DRIVER_BLOCK_HEIGHT-1) in Y direction and (DRIVER_BLOCK_WIDTH-1) in X direction.

window_scroll (x, y);
awindow_scroll (wnd, x, y);

Description:	Scrolls the window x pixels horizontally and y pixels vertically.
Parameters:	**x:** amount of pixels to scroll along horizontal axis
	y: amount of pixels to scroll along vertical axis
Note:	The maximum supported scrolling value is (DRIVER_BLOCK_HEIGHT-1) in Y direction and (DRIVER_BLOCK_WIDTH-1) in X direction.

window_scroll_full (x, y, blk);
awindow_scroll_full (wnd, x, y, blk);

Description:	Scrolls the window x pixels horizontally and y pixels vertically with data transfer.
Parameters:	**x:** amount of pixels to scroll along horizontal axis
	y: amount of pixels to scroll along vertical axis
	blk: the block ID used to fill the blank spaces after tile shifting.

window_adjust_for_background (x, y);
awindow_adjust_for_background (wnd, x, y);

Description: Sets the scroll values for the window with one's complement of x and y.
Note: Useful for rendering a background image with smooth scrolling.
Parameters: **x:** inverted scroll value along horizontal axis
y: inverted scroll value along vertical axis
Example:

```
void RenderMap ( int x, int y )
{
        window_adjust_for_background (x, y);
        background_draw ( x>>3,  y>>3, map, WIDTH, HEIGHT );
}
```

window_no_scroll ( );
awindow_no_scroll (wnd);

Description: Sets both X an Y scrolling values to zero.

window_fix ( );
awindow_fix (wnd);

Description: Adjusts the window to the nearest Standard window - sets the height of it to be a multiply of DRIVER_BLOCK_HEIGHT. The window may be used for scrolling then.

window_set_startline (n);
awindow_set_startline (wnd, n);

Description: Sets the new start video line number in window’s tileset
Parameters: **n:** line number

window_move (n);
awindow_move (wnd, n);

Description: Moves the start video line number in window’s tileset (scrolls up/down)
Parameters: **n:** number of lines (pixels) to move

window_set_height (n);
awindow_set_height (wnd, n);

Description: Sets new height of the window.
Parameters: **n:** height in pixels

window_shrink (n);
awindow_shrink (wnd, n);

Description:	Increases/decreases height of the window.
Parameters:	**n:** signed number of lines to increase/decrease.

2.3 OVERLAY API

2.3.1 Rendering raster bit - maps

```
unsigned char overlay_draw( signed int/unsigned char x,
                            signed int/unsigned char y,
                            PGM_P img,
                            unsigned char width,
                            unsigned char height );
```

Description: Draws a bitmap to the selected context (screen or window).

Parameters:
x: Left-top corner x – coordinate in pixels.
(relative to the left-top corner of the context)
y: Left-top corner y – coordinate in pixels.
(relative to the left-top corner of the context)
img: Points to the bitmap in FLASH.
width: Width of the bitmap in pixels.
height: Height of the bitmap in pixels.

Returns: When the drawing was successfull, the function returns zero.
The error values might be:
1: Out-of-memory. The drawing could not be finished.
2: Vertical coordinate put the whole bitmap outside the canvas.
3: Horizontal coordinate put the whole bitmap outside the canvas.
Other values are reserved for future use.

Notes: When OVERLAY_DRAW_SAFE is enabled, the coords type is *signed int*.
Then the function accepts negative coords and it draws only the positioned bitmap / selected context intersection.
When OVERLAY_INVERSE_PRIORITY is enabled, the latest drawn object has the highest priority (in RAM distribution). So if Out-of-memory condition occurs, the earliest blocks are released and reused for drawing the new object.
When OVERLAY_DRAW_ALPHA is defined, the pixels of the bitmap with color OVERLAY_DRAW_ALPHA will not change the corresponding pixels in the context.
The context (drawing area: screen or window) can be selected in overlay configuration section.
Left / top / right / bottom draw restriction margins can be selected there aswell.

Example

```
void RenderForeground ( int x, int y )
{
        //clear all previously rendered overlay things…
        overlay_clear ( );

        //render the ghost
        overlay_draw ( x, y, SPRITE_GHOST, 20, 20 );
}
```

unsigned char overlay_draw_param(signed int/unsigned char x,
signed int/unsigned char y,
PGM_P img,
unsigned char width,
unsigned char height
unsigned char params);

Description: Draws a bitmap to the selected context with parameters.
The same function as *overlay_draw(…)* but with extra parameters.

Parameters: Same as *overlay_draw(…).*
params: Additional parameters for drawing. The options may be combined with OR.

- Valid when OVERLAY_DRAW_TRANSFORMATIONS enabled:
  OVERLAY_HFLIP – Mirror the source bitmap horizontally.
  OVERLAY_VFLIP – Mirror the source bitmap vertically.
- If OVERLAY_DRAW_COLORMODIFY is enabled, low nibble of the param contains a color value. All the bitmap pixels are E-ORED with this value. So if the low nibble is zero, all pixels remains unchanged.

Returns: Same as *overlay_draw(…).*

Notes: Same as *overlay_draw(…).*
All the *overlay_draw* funtions are available in those two variants (with _param suffix).

overlay_draw_sprite (x, y, sprite);
overlay_draw_sprite_param (x, y, sprite, params);

Description: Draws a sprite (one tile) to the selected context.

Parameters: **x:** Left-top corner x – coordinate in pixels.
(relative to the left-top corner of the context)
y: Left-top corner y – coordinate in pixels.
(relative to the left-top corner of the context)
sprite: Tile ID in tileset of the selected window or default.
(params): See *overlay_draw_param(…).*

Returns: Same as *overlay_draw(…).*

Notes: Same as *overlay_draw(…).*

overlay_draw_sprite_wnd (x, y, sprite, wnd)
overlay_draw_sprite_wnd_param (x, y, sprite, params, wnd)

Description:	Draws a sprite (one tile) from tileset of given window to the selected context.
Parameters:	**x:** Left-top corner x – coordinate in pixels. (relative to the left-top corner of the context) **y:** Left-top corner y – coordinate in pixels. (relative to the left-top corner of the context) **sprite:** Tile ID in tileset of the given window. **wnd:** Handle to a window of which tileset will be used. ***(params):*** See *overlay_draw_param(…).*
Returns:	Same as *overlay_draw(…).*
Notes:	Same as *overlay_draw(…).*

2.3.2 Rendering primitives

unsigned char overlay_putpixel (unsigned char x, unsigned char y, unsigned char color);

Description:	Draws a pixel to the selected context (screen or window).
Parameters:	**x:** Left-top corner x – coordinate in pixels. (relative to the left-top corner of the context) **y:** Left-top corner y – coordinate in pixels. (relative to the left-top corner of the context) **color:** color value for the pixel to be set.
Returns:	When the drawing was successfull, the function returns zero. The error values might be: **1:** Out-of-memory. The drawing could not be finished. Other values are reserved for future use.
Notes:	OVERLAY_PRIMITIVES must be enabled to compile this function.

2.3.3 Managing the overlays

void overlay_clear (void);

Description:	Clears all things rendered with overlay from the screen.
Notes:	OVERLAY_CLEARABLE must be defined to enable this function.

overlay_release ( );

Description: Releases all RAM blocks used by overlay.
Notes: Use this macro instead of slower *overlay_clear(…)* if all the blocks were restored or replaced by user.
OVERLAY_CLEARABLE must be defined to enable this function.

Example:

```
void render()
{
        //assume that the function renderBackground(…) returns true if all
        //backround tiles have been repainted.

        bool res = renderBackground();

        if(res = = TRUE)        overlay_release();
        else                    overlay_clear();

        renderOverlay();
}
```

overlay_get_used_block_count ( );

Description: Returns back amount of RAM blocks currently used for overlay graphics.
Note: The total number of blocks is never greather than OVERLAY_BLOCK_COUNT.

2.4 BACKGROUND API

2.4.1 *Raw reference maps*

void background_draw_simple (unsigned char x, unsigned char y, PGM_P img, unsigned char width, unsigned char height);

Description: Draws a raw (uncompressed) tile reference map to the selected context.
Parameters: **x:** Left-top corner x – coordinate in blocks.
(relative to the left-top corner of the context)
y: Left-top corner y – coordinate in blocks.
(relative to the left-top corner of the context)
img: Points to the tile-reference map in FLASH.
width: Width of the map in blocks.
height: Height of the map in blocks.
Note: Fast, simple version. Does not handle any overflows.
BACKGROUND_DRAW_SIMPLE must be enabled to compile this function.

unsigned char background_draw (signed char x, signed char y, PGM_P img, unsigned char width, unsigned char height);

Description: Draws a raw (uncompressed) tile reference map to the selected context with boundaries overflow handling.

Parameters: **x:** Left-top corner x – coordinate in blocks.
(relative to the left-top corner of the context)
y: Left-top corner y – coordinate in blocks.
(relative to the left-top corner of the context)
img: Points to the tile-reference map in FLASH.
width: Width of the map in blocks.
height: Height of the map in blocks.

Note: Draws only positioned image / selected context intersection. Negative coords are accepted.
BACKGROUND_DRAW must be enabled to compile this function.

2.4.2 Compressed reference maps

void background_draw_RLE (signed char x, signed char y, PGM_P img, unsigned char width, unsigned char height);

Description: Draws a RLE-compressed tile reference map to the selected context with boundaries overflow handling.

Parameters: **x:** Left-top corner x – coordinate in blocks.
(relative to the left-top corner of the context)
y: Left-top corner y – coordinate in blocks.
(relative to the left-top corner of the context)
img: Points to the tile-reference map in FLASH.
width: Width of the map in blocks.
height: Height of the map in blocks.

Note: Draws only positioned image / selected context intersection. Negative coords are accepted. The RLE decompressing macro can be modified in the background.h / config_utils.h.
BACKGROUND_DRAW must be enabled to compile this function.

2.5 MISCELLANEOUS FUNCTIONS

2.5.1 *Delays using frame synchronization*

wait_frames (n);

Description:	Waits for given count of frames.
Parameters:	**n:** Count of frames to wait.
Note:	The sync generator must be running.

wait_seconds (n);

Description:	Waits for given count of seconds.
Parameters:	**n:** Count of seconds to wait.
Note:	The sync generator must be running. Multiplication constant FRAMES_PER_SECOND can be redefined. Default value is 50.

Questions regarding the API description can be discussed in AVGA forums.
Please refer to the home page: http://avga.prometheus4.com.

Jaromir Dvorak, author of the AVGA platform.