PAWS a Risc-V RV32IMAFC CPU – Usage & Programming Guide

About

PAWS is a project to occupy my time in retirement (and as it happened the Covid-19 lockdowns), with the aim of teaching myself about FPGA programming. It is based upon the idea of the 8-bit computers and consoles from the 1980s, but using a modern CPU with C compiler support.

A support library, libPAWS, for easy access to the hardware is provided, along with a few sample C programs to test the hardware and the programming library. This documentation details libPAWS and describes the hardware.

Silice

PAWS is coded in Silice, a hardware description language developed by @sylefeb. Details can be found here [GitHub](https://github.com/sylefeb/Silice) (https://github.com/sylefeb/Silice).

My coding style may not result in the *best* design, the aim was to create a design that could be easily understood.

Using PAWS

The PAWS BIOS showing the SDCARD loader.

The PAWS system starts in the BIOS, which initialises the system, and starts a file explorer for the PAW (compiled programs) files on the SDCARD. Scroll through the available PAW files using LEFT and RIGHT. Use FIRE 1 to select a PAW file, or to enter a directory. Use UP to return from a directory.

If the SDCARD is not detected, try pressing RESET to reinitialise the system.

There are several example programs provided, showing how to use the PAWS graphics system, and other hardware. There are 3 complete games.

| 3DMAZE | ASTROIDS | INVADERS |
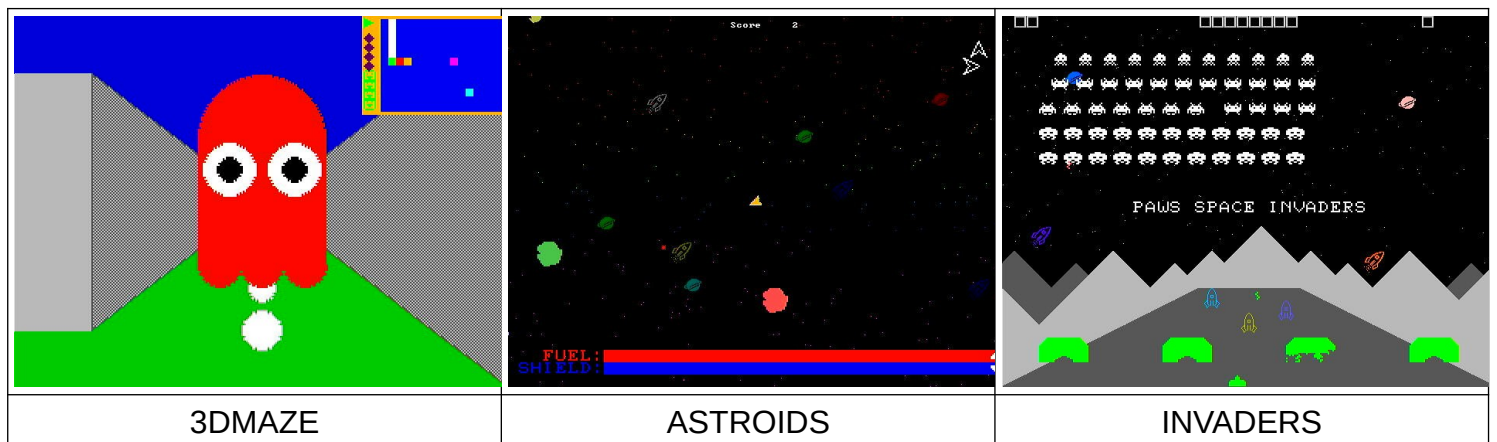
Controlling PAWS via ULX3S Buttons or PS/2 Keyboard

By default, the PS/2 keyboard is mapped as a joystick, with the buttons identified as below.
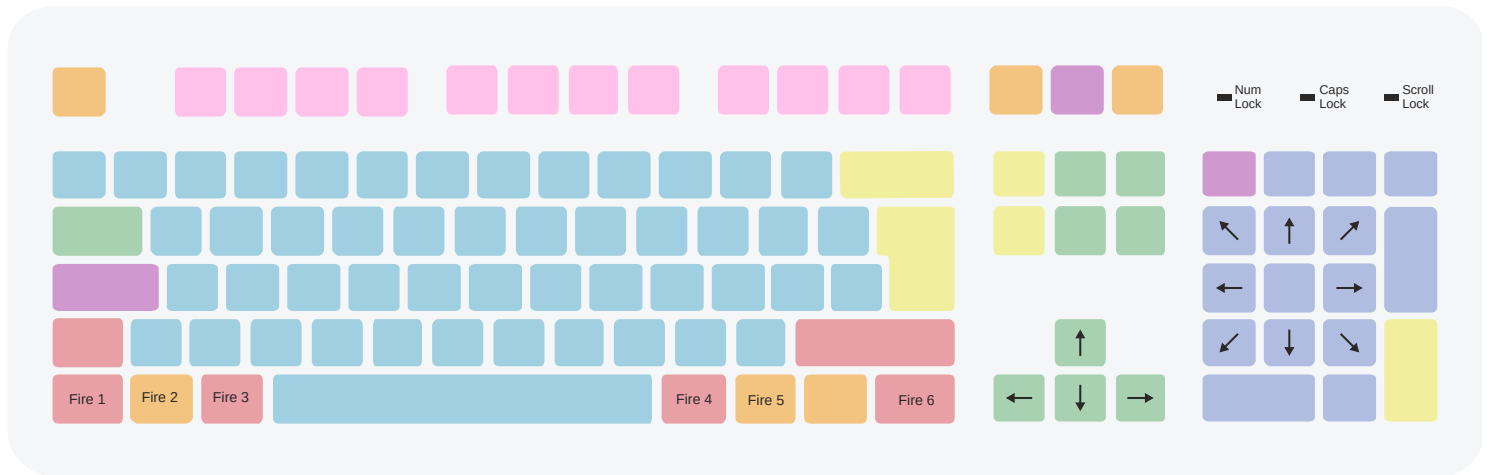
The BIOS can be controlled using either the ULX3S buttons, or the above keyboard keys.

Note: The PS/2 system is not perfect, it does not always reset. It may be necessary to disconnect and reconnect the keyboard for it to reinitialise. I use the CiT KB2106C keyboard which is known to work in PS/2 mode with the ULX3S.

PAWS a Risc-V RV32IMAFC CPU – Usage & Programming Guide
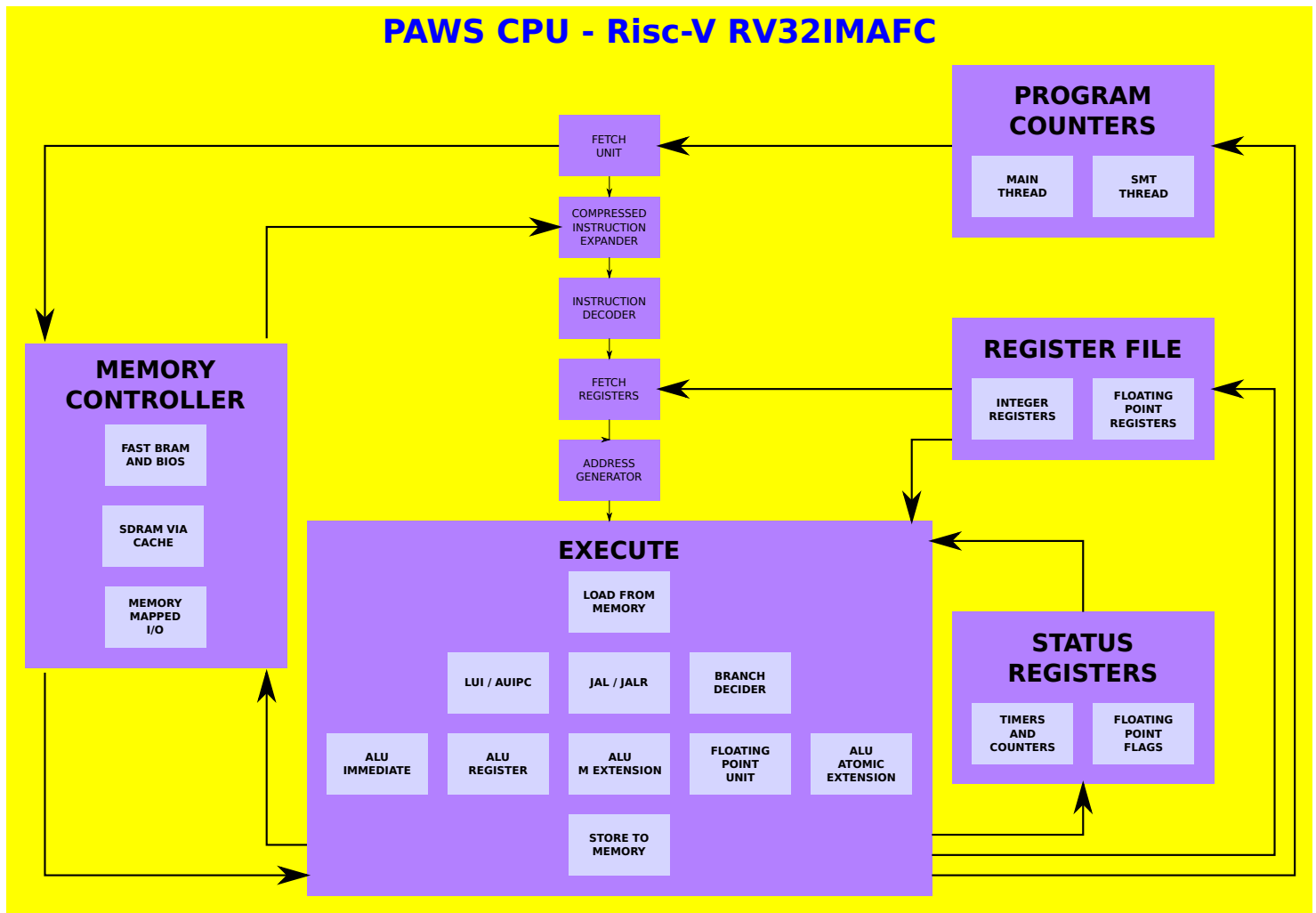
PAWS Features

- Dual Thread Risc-V RV32IMAFC CPU
  - No interrupts
  - No SYSTEM functions or memory protection
- 32MB of SDRAM with CACHE
- 32K of FAST BRAM (used for BIOS and CPU STACKS)
- 32K of I/O MEMORY MAPPED REGISTERS

- PS/2 Keyboard
  - Joystick emulation mode available
- 115200 baud UART
- ULX3S Buttons and LEDS
- SDCARD with FAT32 (READ ONLY)
- 1hz and 1khz Timers
- Pseudo Random Number Generators
- Simple Stereo Audio
  - Square, Sawtooth, Triangle and Sine Waves, plus Noise

- Multi-Layered Display (64 colours where available , with transparency)
  - TERMINAL
    - 80 x 8 WHITE on BLUE
  - CHARACTER MAP
    - 80 x 60 COLOUR, NORMAL and BOLD FONTS
  - SPRITES
    - 2 LAYERS OF 16 16x16 SINGLE COLOUR SPRITES
  - TILEMAP
    - 2 LAYERS OF 42x32 16x16 TILES (40 x 30 DISPLAYED) with SCROLLING
  - BITMAP
    - DOUBLE BUFFERED 320 x 240 COLOUR BITMAP
      - HARDWARE ASSISTED DRAWING OF
        - Points
        - Lines
        - Filled Rectangles
        - Circles, Filled and Outline
        - Filled Triangles
        - Single colour and colour blitters
        - Vectors
    - PROGRAMMABLE BACKGROUND DISPLAY

PAWS CPU and SOC

The PAWS CPU is a Risc-V RV32IMAFC that implements only the features needed to run GCC or LLVM/CLANG compiled code.

- No interrupts.
- Machine mode only.

The PAWS CPU has two modes:

- Single thread using all available cycles.
- Dual thread
  - Execute an instruction from each thread alternatively.
  - Second thread can be stopped/started as required.

PAWS a Risc-V RV32IMAFC CPU – Usage & Programming Guide

Supported Instruction Set Listing

Risc-V	Implemented	Not Implemented	Notes
BASE	ADD[I] SUB SLT[I][U] AND[I] OR[I] XOR[I] SLL[I] SRL[I] SRA[I] AUIPC/LUI		
BASE unconditional jumps	JAL[R}		
BASE conditional brnaches	BEQ/BNE BLT[U] BGE[U]		
BASE load and store	LB[U] LH[U] LW SB SH SW		
BASE		FENCE FENCE.I	
BASE and F EXTENSION CSR	RDCYCLE[H] RDTIME[H] RDINSTRET[H] F[R][S]CSR F[R][S]RM F[R][S]FLAGS FS[RM][FLAGS]I		Timers and instruction retired counters are readonly.
BASE SYSTEM		ECALL EBREAK	
M EXTENSION	DIV[U] REM[U] MUL MULH[[S]U]		
A EXTENSION	AMOADD AMOSWAP AMOAND AMOOR AMOXOR AMOMAX[U] AMOMIN[U]		AQ / RL flags are ignored. The AMO instructions do operate as a complete READ-MODIFY_WRITE operation, as intended.
F EXTENSION	FLW FSW F[N]M[ADD][SUB].S FADD.S FSUB.S FMUL.S FDIV.S FSQRT.S FSNJ[N][X].S FMIN.S FMAX.S FCVT.W[U].S FCVT.S.W[U] FMV.X.W FMV.W.X FEQ.S FLT.S FLE.S FCLASS.S		There is no rounding control.

PAWS a Risc-V RV32IMAFC CPU – Usage & Programming Guide

Compiling Programs For PAWS

The default language for PAWS is C, specifically GCC, with support for LLVM and CLANG.

To create a program for PAWS, create a C file in the SOFTWARE/c directory. It is advised to use the SOFTWARE/template.c as a starting point.

Contents of template.c	Explanation
`#include "PAWSlibrary.h"`	Use libPAWS for definitions and helper functions.

`int main( void ) {`
`    INITIALISEMEMORY();`
 	
`    // CODE GOES HERE`
`}`
 	
`// EXIT WILL RETURN TO BIOS`
	MAIN program entry point
	Setup the memory map
	
	Main loop.

Compile your code using the shell scripts. For example, to compile the included asteroids style arcade game, `./compile.sh c/asteroids.c PAWS/ASTROIDS.PAW` or `./clang.sh c/asteroids.c PAWS/ASTROIDS.PAW.` `Either` will compile the program to PAWS/ASTROIDS.PAW, which can be copied to the SDCARD for loading via the BIOS.

PAWS uses newlib to provide a C library and some auxiliary floating-point routines, and libgcc to provide additional floating-point routines (such as single precision to double precision). The shell scripts will link to these libraries installed in their default locations on ArchLinux.

PAWS a Risc-V RV32IMAFC CPU – Usage & Programming Guide

PAWS Memory System

The SDRAM has a 16k directly mapped eviction cache. A directly mapped cache was used to simplify the cache logic, and an eviction method chosen to reduce the number of SDRAM writes.

Memory access are organised as 16 bit, the bus width of the SDRAM chip on the ULX3S. 16-bit compressed instructions are preferred as due to latency during instruction fetching these will be fetched, decoded and executed quicker than 32-bit instructions.

Memory Management

The BIOS will initialise the memory, and allocates space at the top of fast BRAM memory for the CPU STACKS (MAIN and SMT THREADS).

Address Range	Memory Type	Usage
0x00000000 – 0x00008000	Fast BRAM	0x0000 – 0x1000 BIOS 0x8000 – 0x4000 Main Stack 0x4000 – 0x2000 SMT Stack 0x1000 – 0x1400 printf buffer 0x1400 – 0x1500 libPAWS 0x1500 – 0x2000 fast storage
0x00008000 – 0x0000ffff	I/O Registers	Commuincation with the PAWS hardware. No direct hardware access is required, as libPAWS provides functions for all aspects of the PAWS hardware.
0x4000000 - 0x7ffffff	SDRAM PROGRAM + LOADED DATA MALLOC ALLOCATED MEMORY	Program and data storage. Accessed via a cache.

libPAWS variables and functions

unsigned char *MEMORYTOP	Points to the top of unallocated memory.
void INITIALISEMEMORY(void)	Sets up the memory map using parameters passed from the BIOS.

Standard C library functions for memory management such as malloc are available via newlib. See newlib documentation for details.

Colours

PAWS uses a 6-bit colour attribute, given as RRGGBB. This gives 64 colours, specified in hexadecimal as per the table below. Names defined in libPAWS are shown.

BLACK	0x01	DKBLUE	BLUE	0x04	0x05	0x06	LTBLUE
DKGREEN	0x09	0x0a	DKCYAN	GREEN	0x0d	0x0e	CYAN
0x10	DKPURPLE	0x12	PURPLE	0x14	GREY1	0x16	LTPURPLE
0x18	0x19	0x1a	0x1b	0x1c	LTGREEN	0x1e	LTCYAN
DKRED	0x21	DKMAGENTA	0x23	BROWN	0x25	0x26	0x27
DKYELLOW	0x29	GREY2	0x2b	0x2c	0x2d	0x2e	0x2f
RED	0x31	0x32	MAGENTA	ORANGE	LTRED	0x36	LTMAGENTA
ORANGE	LTORANGE	PEACH	PINK	YELLOW	LTYELLOW	0x3e	WHITE

Some display layers allow for a transparency attribute to allow lower layers to show. This is named TRANSPARENT in libPAWS.

Display Structure

The display in PAWS is organised in layers. The arrangement of the layers can be adjusted, with the background layer always being at the bottom.

The default arrangement of layers (top to bottom) is:

- Terminal Layer (hidden by default)
- Character (Text) Layer
- Upper Sprite Layer
- Bitmap Layer
- Lower Sprite Layer
- Upper Tile Map Layer
- Lower Tile Map Layer
- Background Layer

PAWS Asteroids, showing the background (dark blue and falling stars), the bitmap (logo, galaxy image, "GAME OVER" and the fuel bars), the tile maps (the small planets and rocket ships), the sprites (asteroids, UFO, player ship), and the character map (player score and instructions).

PAWS Asteroids runs in screen mode 2, where the bitmap is displayed below the sprites and the tile maps.

libPAWS Variables and Functions

`void await_vblank( void )`	Waits for the screen vertical blank to start.
`void screen_mode( unsigned char screenmode, unsigned char colour )`	Changes the display layer order and selects colour or greyscale mode.

File Management

libPAWS has the ability to load files from the SDCARD directly into memory.

libPAWS variables and functions

unsigned char *sdcard_selectfile(char *message, char *extension, unsigned int *filesize)	Returns a pointer to a file that has been loaded into memory, or NULL if no file found.
	Message is displayed above the file selector. Only directories and files of the type "extension" will be displayed.

Example code for loading a JPG into memory via the file selector, decoding and displaying.

```
#include "PAWSlibrary.h"
#include <stdlib.h>

int main( void ) {
    INITIALISEMEMORY();

    int width, height; unsigned int filesize;
    unsigned char *imagebuffer, colour, *filebuffer;

    filebuffer = sdcard_selectfile( "Please select a JPEG", "JPG", &filesize );

    // JPEG LIBRARY
    if( filebuffer ) {
        njInit();
        njDecode( filebuffer, filesize );
        width = njGetWidth();
        height = njGetHeight();
        imagebuffer=njGetImage();
        gpu_pixelblock24( 0, 0, width, height, imagebuffer );
        free( filebuffer );
    } else {
        gpu_print_centre( WHITE, 160, 120, 0, 0, "NO FILE FOUND!" );
    }

    sleep( 4000, 0 );
}
```

PAWS a Risc-V RV32IMAFC CPU – Usage & Programming Guide

Single Thread or Dual Thread Mode

On startup PAWS runs in single thread mode. The BIOS will switch back to single thread mode when returning to the BIOS from a program running in dual thread mode.

Whilst dual thread mode is activated PAWS will execute one instruction from each thread alternatively. All memory is shared, with no memory protection.

The Risc-V A Extension (Atomic Instructions) are decoded and executed, but ignoring the *aq* and *rl* flags. The whole of the fetch-modify-write cycle will complete before allowing the other thread to execute. FENCE instructions from the Risc-V base are treated as no-ops.

libPAWS variables and functions

void SMTSTOP(void)void SMTSTART(unsigned int code)	Stops the SMT thread.
void SMTSTART(unsigned int code)	Starts the SMT thread, jumping immediately to the address of the function provided.

Due to the way that all of the I/O operations are memory mapped there are considerations to make when writing dual threaded code. Only one thread should attempt to use a section of the graphics system.
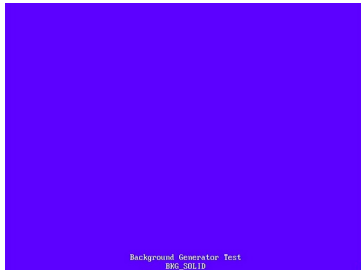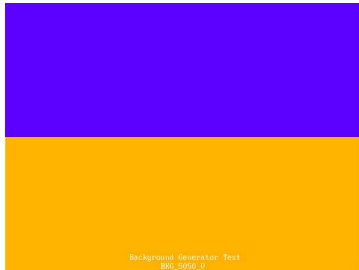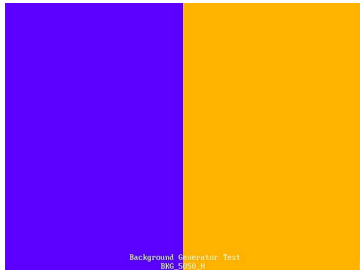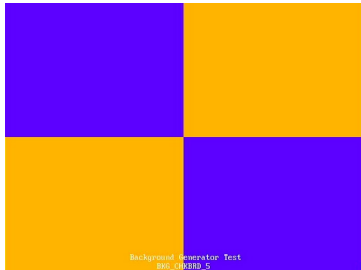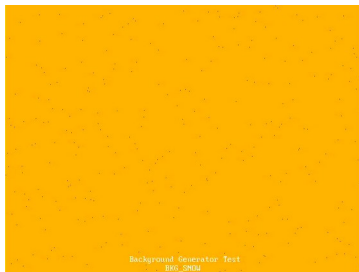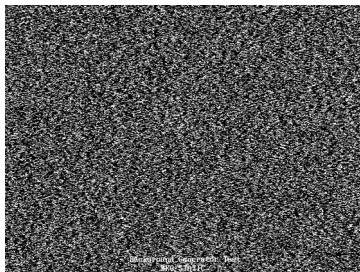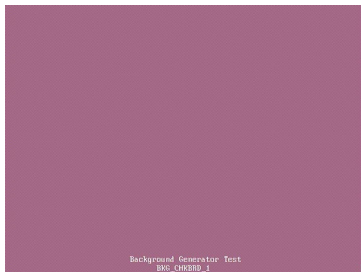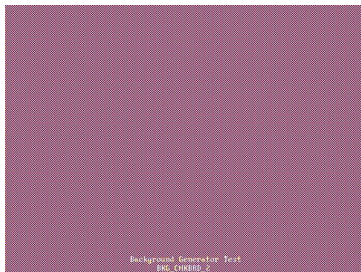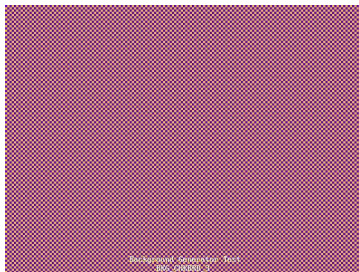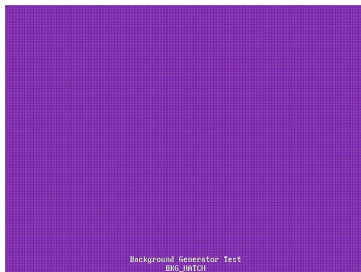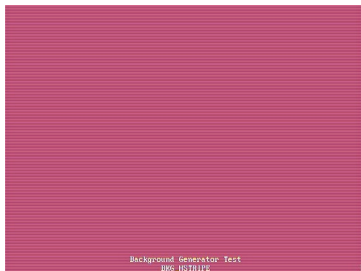
```
Example code for a simple dual thread program

#include "PAWSlibrary.h"

void smtthread( void ) {
    // SETUP STACKPOINTER FOR THE SMT THREAD
    asm volatile ("li sp ,0x4000");
    while(1) {
        gpu_rectangle( rng( 64 ), rng( 640 ), rng( 432 ), rng( 640 ), rng( 432 ) );
        sleep( 500, 1 );
    }
}

void main( void ) {
    INITIALISEMEMORY();

    tpu_printf_centre( 27, TRANSPARENT, GREEN, "SMT Test" );
    tpu_printf_centre( 28, TRANSPARENT, YELLOW, "I'm Just Sitting Here Doing Nothing" );
    tpu_printf_centre( 29, TRANSPARENT, BLUE, "The SMT Thread Is Drawing Rectangles!" );
    SMTSTART( (unsigned int )smtthread );

    while(1) {
        tpu_set( 1, 1, TRANSPARENT, WHITE );
        tpu_printf( "Main Thread Counting Away: %d", systemclock() );
        sleep( 1000, 0 );
    }
}
```
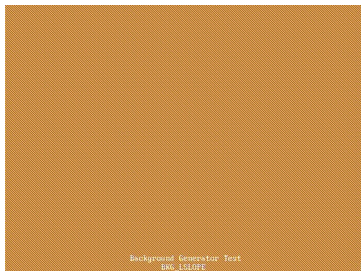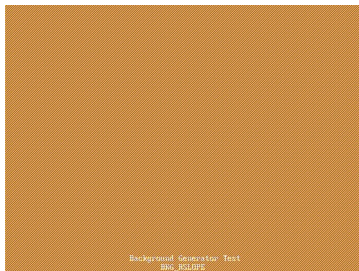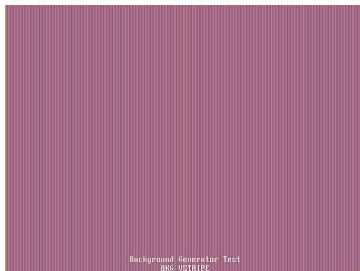
NOTE: The first line of code in the smtthread function **must** set the stack pointer to the reserved memory in the fast BRAM.

PAWS a Risc-V RV32IMAFC CPU – Usage & Programming Guide

Background Generator

The background layer shows when there is nothing to display from the layers above. There are 16 named background generators in libPAWS. The result of the individual background generators are shown in the table below.

Result of set_background(PURPLE, ORANGE, *value from table*);			
BKG_SOLID	BKG_5050_V	BKG_5050_H	BKG_CHKBRD_5
BKG_RAINBOW	BKG_SNOW	BKG_STATIC	BKG_CHKBRD_1
BKG_CHKBRD_2	BKG_CHKBRD_3	BKG_CHKBRD_4	BKG_HATCH
BKG_LSLOPE	BKG_RSLOPE	BKG_VSTRIPE	BKG_HSTRIPE

In addition, a simple co-processor, called COPPER, is available to change background generator parameters during the frame generation.

Background COPPER Programming

The COPPER is designed to allow the changing of the background generator parameters during the display frame. The COPPER program storage has 64 entries, there is one variable, and it can detect the present X and Y coordinates (range X = 0 to 639, Y = 0 to 479).

COMMAND	CONDITION	VALUE	MODE	ALT	COLOUR / ADDRESS	
JUMP	ALWAYS				ADDRESS	Jump to ADDRESS
JUMP	IF_VBLANK_EQUAL	0 or 1			ADDRESS	Jump to ADDRESS if VBLANK is 0 or 1
JUMP	IF_HBLANK_EQUAL	0 or 1			ADDRESS	Jump to ADDRESS if HBLANK is 0 or 1
JUMP	IF_Y_LESS	Y COORDINATE*			ADDRESS	Jump to ADDRESS if Y is LESS THAN VALUE
JUMP	IF_X_LESS	X COORDINATE*			ADDRESS	Jump to ADDRESS if X is LESS THAN VALUE
JUMP	IF_VARIABLE_LESS	VALUE*			ADDRESS	Jump to ADDRESS if VARIABLE is LESS THAN VALUE
WAIT_VBLANK	SET FLAGS		MODE	ALT	COLOUR	Wait for VBLANK and SET
WAIT_HBLANK	SET FLAGS		MODE	ALT	COLOUR	Wait for HBLANK and SET
WAIT_Y	SET FLAGS	Y COORDINATE*	MODE	ALT	COLOUR	Wait for Y and SET
WAIT_X	SET FLAGS	X COORDINATE*	MODE	ALT	COLOUR	Wait for X and SET
WAIT_VARIABLE	SET FLAGS	X/Y FLAG	MODE	ALT	COLOUR	Wait for VARIABLE to be EQUAL to X or Y
SET_VARIABLE	1	VALUE*				Set VARIABLE to VALUE
ADD_VARIABLE	2	VALUE*				Add VALUE to VARIABLE
SUB_VARIABLE	4	VALUE*				Subtract VALUE from VARIABLE
SET_FROM_VARIABLE	SET_FLAGS					SET from VARIABLE
* the value can be replaced with the constant 'COPPER_USE_CPU_INPUT' which will then compare/use the value set by the libPAWS function set_copper_cpuinput(value).						

A simple COPPER program that sets the background generator to the BKG_SNOW pattern on a BLACK background, and changes the colour of the snow/stars every 64 pixels down the screen to give a rainbow effect.

```
    copper_startstop( 0 );
    copper_program( 0, COPPER_WAIT_Y, 7, 0, BKG_SNOW, BLACK, WHITE );
    copper_program( 1, COPPER_WAIT_X, 7, 0, BKG_SNOW, BLACK, WHITE );
    copper_program( 2, COPPER_JUMP, COPPER_JUMP_IF_Y_LESS, 64, 0, 0, 1 );
    copper_program( 3, COPPER_WAIT_X, 7, 0, BKG_SNOW, BLACK, RED );
    copper_program( 4, COPPER_JUMP, COPPER_JUMP_IF_Y_LESS, 128, 0, 0, 3 );
    copper_program( 5, COPPER_WAIT_X, 7, 0, BKG_SNOW, BLACK, ORANGE );
    copper_program( 6, COPPER_JUMP, COPPER_JUMP_IF_Y_LESS, 160, 0, 0, 5 );
    copper_program( 7, COPPER_WAIT_X, 7, 0, BKG_SNOW, BLACK, YELLOW );
    copper_program( 8, COPPER_JUMP, COPPER_JUMP_IF_Y_LESS, 192, 0, 0, 7 );
    copper_program( 9, COPPER_WAIT_X, 7, 0, BKG_SNOW, BLACK, GREEN );
    copper_program( 10, COPPER_JUMP, COPPER_JUMP_IF_Y_LESS, 224, 0, 0, 9 );
    copper_program( 11, COPPER_WAIT_X, 7, 0, BKG_SNOW, BLACK, LTBLUE );
    copper_program( 12, COPPER_JUMP, COPPER_JUMP_IF_Y_LESS, 256, 0, 0, 11 );
    copper_program( 13, COPPER_WAIT_X, 7, 0, BKG_SNOW, BLACK, PURPLE );
    copper_program( 14, COPPER_JUMP, COPPER_JUMP_IF_Y_LESS, 288, 0, 0, 13 );
    copper_program( 15, COPPER_WAIT_X, 7, 0, BKG_SNOW, BLACK, MAGENTA );
    copper_program( 16, COPPER_JUMP, COPPER_JUMP_IF_NOT_VBLANK, 0, 0, 0, 15 );
    copper_program( 17, COPPER_JUMP, COPPER_JUMP_ALWAYS, 0, 0, 0, 1 );
    copper_startstop( 1 );
```

A simple COPPER program that sets the background generator to the BKG_SNOW pattern on a BLACK background, and cycles through the colours for the snow/stars every frame, giving a twinkling stars effect.

```
    copper_startstop( 0 );
    copper_program( 0, COPPER_VARIABLE, COPPER_SET_VARIABLE, 1, 0, 0, 0 );
    copper_program( 1, COPPER_WAIT_Y, 6, 0, BKG_SNOW, BLACK, 0 );
    copper_program( 2, COPPER_SET_FROM_VARIABLE, 1, 0, 0, 0, 0 );
    copper_program( 3, COPPER_VARIABLE, COPPER_ADD_VARIABLE, 1, 0, 0, 0 );
```

```
copper_program( 4, COPPER_JUMP, COPPER_JUMP_IF_NOT_VBLANK, 0, 0, 0, 4 );
copper_program( 5, COPPER_JUMP, COPPER_JUMP_IF_VARIABLE_LESS, 64, 0, 0, 1 );
copper_program( 6, COPPER_JUMP, COPPER_JUMP_ALWAYS, 0, 0, 0, 0 );
copper_startstop( 1 );
```

PAWS a Risc-V RV32IMAFC CPU – Usage & Programming Guide

Bitmap and GPU

PAWS displays a 320x240 pixel bitmap with 64 colours, plus TRANSPARENT. PAWS uses a GPU to draw to the bitmap, with hardware accelerated drawing of some graphic primitives.

Basic Principles

The bitmap is 320 pixels wide by 240 pixels tall, with (0,0) being located at the top left-hand corner of the screen. Drawing can be limited to a "cropping rectangle", only pixels within the "cropping rectangle" will be drawn.

`void bitmap_display( unsigned char framebuffer )`	Display bitmap 0 or 1.
`void bitmap_draw( unsigned char framebuffer )`	Draw to bitmap 0 or 1.
`void gpu_crop( unsigned short left, unsigned short right, unsigned short top, unsigned short bottom )`	Sets the cropping rectangle for the GPU. gpu_crop(CROPFULLSCREEN) will return to the whole of the screen.
`void gpu_cs( void )`	Clears the bitmap to TRANSPARENT.

Example code for a tear-free animation
| ```
#include "PAWSlibrary.h"

void main( void ) {
    // CURRENT FRAMEBUFFER
    unsigned short framebuffer = 0;

    INITIALISEMEMORY();

    while(1) {
        // DRAW TO HIDDEN BITMAP
        bitmap_draw( !framebuffer );

        // CODE TO GENERATE THE BITMAP
        // DRAWN TO THE HIDDEN BITMAP

        // SWITCH THE FRAMEBUFFER
        await_vblank();
        framebuffer = !framebuffer;
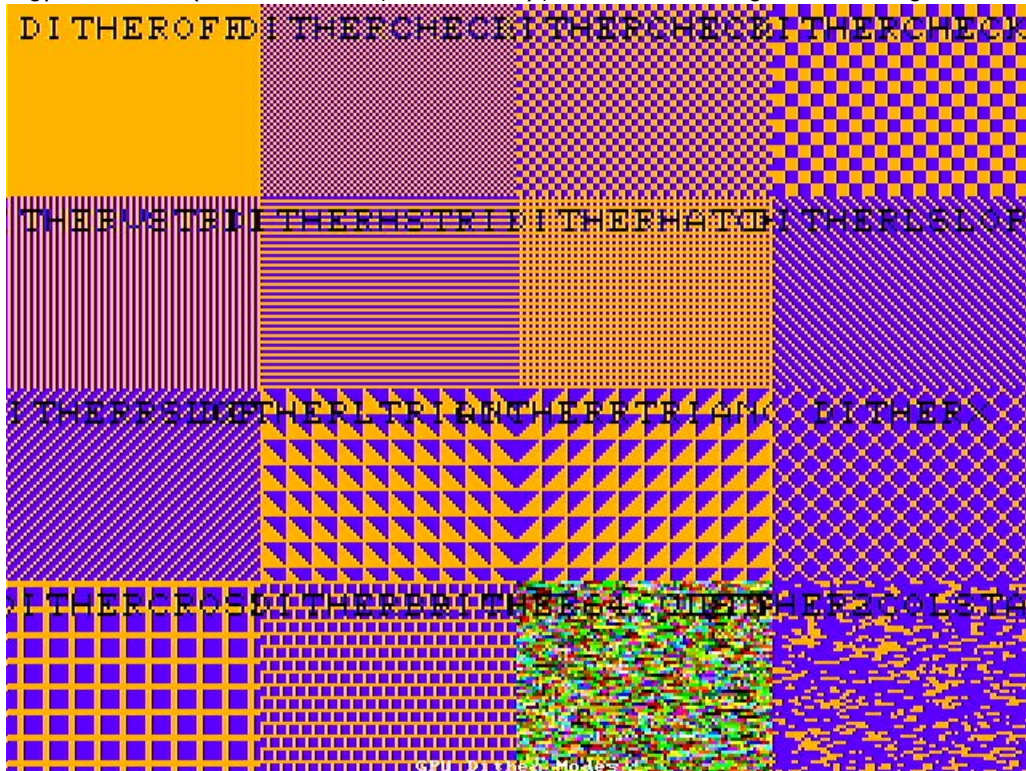        bitmap_display( framebuffer );
    }
}
``` |

Dither Patterns

When drawing rectangles, filled circles or triangles, the GPU can apply one of 16 "dither" patterns. The "dither" routine determines whether to use the main drawing colour, provided with the shape being drawn, or the alternate colour, set with the "dither mode".

`void gpu_dither( unsigned char mode, unsigned char colour )`	Sets the dither mode and the alternate colour.

Result of `gpu_dither( dithermode, PURPLE );` Plus drawing a rectangle in ORANGE.

Graphic Primatives

`void gpu_pixel( unsigned char colour, short x, short y )`	Draws a pixel at (x,y) in colour.
`void gpu_line( unsigned char colour, short x1, short y1, short x2, short y2 )`	Draws a line from (x1,y1) to (x2,y2) in colour.
`void gpu_wideline( unsigned char colour, short x1, short y1, short x2, short y2, unsigned short width )`	Draws a line from (x1,y1) to (x2,y2) in colour, width pixels wide. Effectively draws a parallelogram. Steep lines with the flat sides horizontal, shallow lines with the flat sides vertical.
`void gpu_rectangle( unsigned char colour, short x1, short y1, short x2, short y2 )`	Draws a filled rectangle with corners at (x1,y1) and (x2,y2) in colour. Uses the dither mode.
`void gpu_circle( unsigned char colour, short x1, short y1, short radius, unsigned char drawsectors, unsigned char filled )`	Draws a circle at centre (x1,y1) of the given radius in colour, optionally filled. Uses the dither mode when filling. See below for details of the drawsectors parameter.
`void gpu_triangle( unsigned char colour, short x1, short y1, short x2, short y2, short x3, short y3 )`	Draws a filled triangle with vertices (x1,y1), (x2,y2) and (x3,y3) in colour. Uses the dither mode.
`void gpu_quadrilateral( unsigned char colour, short x1, short y1, short x2, short y2, short x3, short y3, short x4, short y4 )`	Draws a filled **convex** quadrilateral with vertices (x1,y1), (x2,y2), (x3,y3) and (x4,y4) in colour. Uses the dither mode. NOTE: Drawn by breaking into two triangles, so vertices should be presented either anti-clockwise or clockwise.

The drawsectors parameter is an 8-bit binary mask that specifies which of the 45° sectors will be drawn. The bit position that represents each sector is as shown.

Blitters

The blitter copies a tile to the bitmap, drawing pixels where there is a set bit in the tile, and ignoring clear bits in the tile (treating them as transparent). Pixels are drawn in the specified colour without dither patterns being applied. The colour blitter does the same, except that the tile describes a 64 colour image, with transparent pixels being ignored.

The blit_size parameter specifies the size of the output.

blit_size	Size of gpu_blit and gpu_colourblit output	Size of gpu_character_blit
0	16x16	8x8
1	32x32	16x16
2	64x64	32x32
3	128x128	64x64

The action parameter specifies the reflection or rotation of the tile when draw to the screen.

action	Result
0	No effect (no reflection)
1 (REFLECT_X)	Reflect in X axis
2 (REFLECT_Y)	Reflect in Y axis
3 (REFLECT_X | REFLECT_Y)	Reflect in X and Y axes
4 (ROTATE0)	No effect (rotate 0 degrees)
5 (ROTATE90)	Rotate 90 degrees anti-clockwise
6 (ROTATE180)	Rotate 180 degrees anti-clockwise
7 (ROTATE270)	Rotate 270 degrees anti-clockwise

Tile Blitter

The tile blitter uses 16x16 tiles, 64 are available, with the tiles being set using the set_blitter_bitmap function, which takes a pointer to an array of 16 unsigned short.

`void gpu_blit( unsigned char colour, short x1, short y1, short tile, unsigned char blit_size, unsigned char action )`	Blit a 16x16 tile to (x1,y1) in colour.
`void set_blitter_bitmap( unsigned char tile, unsigned short *bitmap )`	Define one of the 64 16x16 blitter tiles.

Code	Output

`unsigned short blitter_bitmaps[] = {`
`    0b0000011000000000,`
`    0b0000111100000000,`
`    0b0001111110000000,`
`    0b0011011011000000,`
`    0b0011111111000000,`
`    0b0001011010000000,`
`    0b0010000001000000,`
`    0b0001000010000000,`
`    0,0,0,0,0,0,0,0 };`
 	
`set_blitter_bitmap( 0, &blitter_bitmaps[0] );`
 	
`gpu_blit( GREEN, 16, 32, 0, 0, 0 );`
	(16,32)

The character blitter uses 8x8 tiles, 512 are available, with the tiles being set using the set_blitter_chbitmap function, which takes a pointer to an array of 8 unsigned char. The character blitter tiles default to an 8x8 character ROM, with normal (characters 0 to 255) and bold characters (characters 256 to 511), but can be overwritten.

The colour blitter uses 16x16 tiles, 64 are available, with the tiles being set using the set_colourblitter_bitmap function, which takes a pointer to an array of 256 unsigned char.

`void gpu_character_blit( unsigned char colour, short x1, short y1, unsigned char tile, unsigned char blit_size, unsigned char action )`	Blit an 8x8 character to (x1,y1) in colour.
`void gpu_colourblit( short x1, short y1, short tile, unsigned char blit_size, unsigned char action )`	Blit a 16x16 full colour tile to (x1,y1).

`void set_blitter_chbitmap( unsigned char tile, unsigned char *bitmap )`	Define one of the 256 8x8 character blitter tiles.
`void set_colourblitter_bitmap( unsigned char tile, unsigned char *bitmap )`	Define one of the 64 16x16 colour blitter tiles.

Pixelblocks

The pixelblock operates as a software blitter, outputting pixels from an area of memory, such as a decoded image. gpu_pixelblock7 outputs { ARRGGBB } pixels, and gpu_pixelblock24 converts 24bit RGB pixels to the PAWS ARRGGBB pixels before output.

The transparent parameter specifies a colour to treat as a 'mask' and not to be drawn. If ALL pixels are to be output, set transparent to TRANSPARENT.

`void gpu_pixelblock7( short x, short y, unsigned short w, unsigned short h, unsigned char transparent. Unsigned char *buffer )`	Outputs a rectangle of { ARRGGBB } pixels stored in memory, starting at (x,y) of size (width, height).
`void gpu_pixelblock24( short x, short y, unsigned short width, unsigned short height, unsigned char transparent. Unsigned char *buffer )`	Outputs a rectangle of 24bit RGB pixels stored in memory, starting at (x,y) of size (width, height).

The pixelblock can also operate by taking pixels one-by-one, such as after calculation and drawing to the bitmap.

This mode is activated using gpu_pixelblock_start, and terminated with gpu_pixelblock_stop. Pixels are sent using gpu_pixelblock_pixel7 and gpu_pixelblock_pixel24 for { ARRGGBB } and 24-bit RGB pixels respectively.

`void gpu_pixelblock_start( short x, short y, unsigned short width )`	Set the GPU to start accpeting a rectangle of width pixels starting at (x,y) in PIXELBLOCK mode. NOTE: No other GPU commands can be issued unitl gpu_pixelblock_stop() has been called.
`void gpu_pixelblock_stop( void )`	Stop the PIXELBLOCK mode.
`void gpu_pixelblock_pixel7( unsigned char pixel )`	Send an { ARRGGBB } pixel to the pixelblock and move to the next pixel.
`void gpu_pixelblock_pixel24( unsigned char red, unsigned char green, unsigned char blue )`	Send a 24bit RGB pixel to the pixelblock and move to the next pixel.

NOTE: No other GPU operation can be started until gpu_pixelblock_stop has been called.

Miscellaneous/Utility Functions

`void gpu_box( unsigned char colour, short x1, short y1, short x2, short y2, unsigned short width )`	Draws an outline rectangle with corners at (x1,y1) and (x2,y2) in colour, width pixels wide. NOTE: Drawn by breaking into 4 lines.

To print C formatted strings to the bitmap, using the character blitter to output each character.

For gpu_printf the coordinate specify the top-left pixel for the output; for gpu_printf centre the coordinate specifies the top and the horizontal centre pixel for the output; for gpu_printf_vertical the coordinate specifies the bottom-left pixel for the output; for gpu_printf_centre_vertical the coordinate specifies the left and the vertical centre pixel for the output.

The size and action parameters are obeyed as per the character blitter.

`void gpu_printf( unsigned char colour, short x, short y, unsigned char size, unsgined char action, const char *fmt,... )`	Outputs a string (maximum 80 characters) by repeatedly using the character blitter, starting at (x,y) in colour. Will size the characters and space accordingly. Action is detailed below. NOTE: Escape characters are not processed, the corresponding character code is output as a character.
`void gpu_printf_vertical( unsigned char colour, short x, short y, unsigned char size, unsgined char action, const char *fmt,... )`	Outputs a string (maximum 80 characters) by repeatedly using the character blitter, starting at (x,y) in colour. Will size the characters and space accordingly. Moving vertically upwards. Action is detailed below. NOTE: Escape characters are not processed, the corresponding character code is output as a character.
`void gpu_printf_centre( unsigned char colour, short x, short y, unsigned char size, unsigned char action, const char *fmt, ... )`	Outputs a string by repeatedly using the character blitter, with the top centred at (x,y) in colour. Will size the characters and space accordingly. Action is detailed below. NOTE: Escape characters are not processed, the corresponding character code is output as a character.
`void gpu_printf_centre_vertical( unsigned char colour, short x, short y, unsigned char size, unsigned char action, const char *fmt, ... )`	Outputs a string by repeatedly using the character blitter, with the top centred at (x,y) in colour. Will size the characters and space accordingly. Moving vertically upwards. Action is detailed below. NOTE: Escape characters are not processed, the corresponding character code is output as a character.

Hardware Vector Blocks

`void draw_vector_block( unsigned char block, unsigned char colour, short xc, short yc, unsigned char rotation )`	Starts the drawing of one of the 32 user definable vector (line drawn) objects, centred at (x,y) in colour. Rotation is by rotation*90°.
`void set_vector_vertex( unsigned char block, unsigned char vertex, unsigned char active, char deltax, char deltay )`	Sets one of the 16 vertices in one of the 32 user definable vector (line drawn) objects.

The hardware vector block draws shapes that are defined by up to 15 lines, defined by 16 vertices. The lines are drawn in dot-to-dot fashion; the first line is defined by vertices 0 and 1; the second line is defined by vertices 1 and 2; until an inactive vertex is arrived at, or the last vertex is used.

The vertices are defined as deltas from (0,0) at the origin, with a range of -31 to 31.

From the asteroids.c code to setup the ship shape to draw the remaining lives.

```
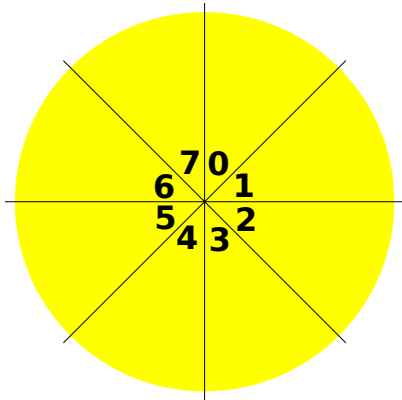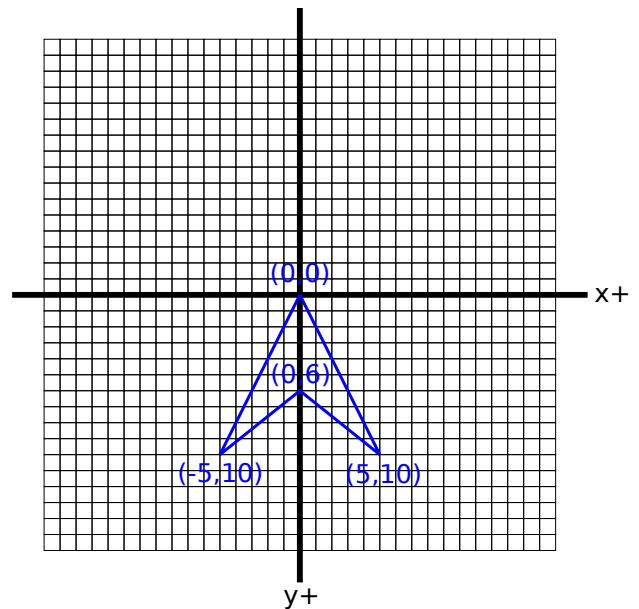void set_ship_vector( void ) {
    set_vector_vertex( 0, 0, 1, 0, 0 );
    set_vector_vertex( 0, 1, 1, 5, 10 );
    set_vector_vertex( 0, 2, 1, 0, 6 );
    set_vector_vertex( 0, 3, 1, -5, 10 );
    set_vector_vertex( 0, 4, 1, 0, 0 );
    set_vector_vertex( 0, 5, 0, 0, 0 );
}
```

Note that the deltas work the same as the bitmap coordinates, positive x is to the right, positive y to the bottom.

Note that the last vertex is set to active=0 so as to stop the vector drawer processing any further vertices in this block.

Tilemaps

Sprites

Text

Terminal