PAWS a Risc-V RV32IMA[F]C[B] CPU – 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.

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).

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

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

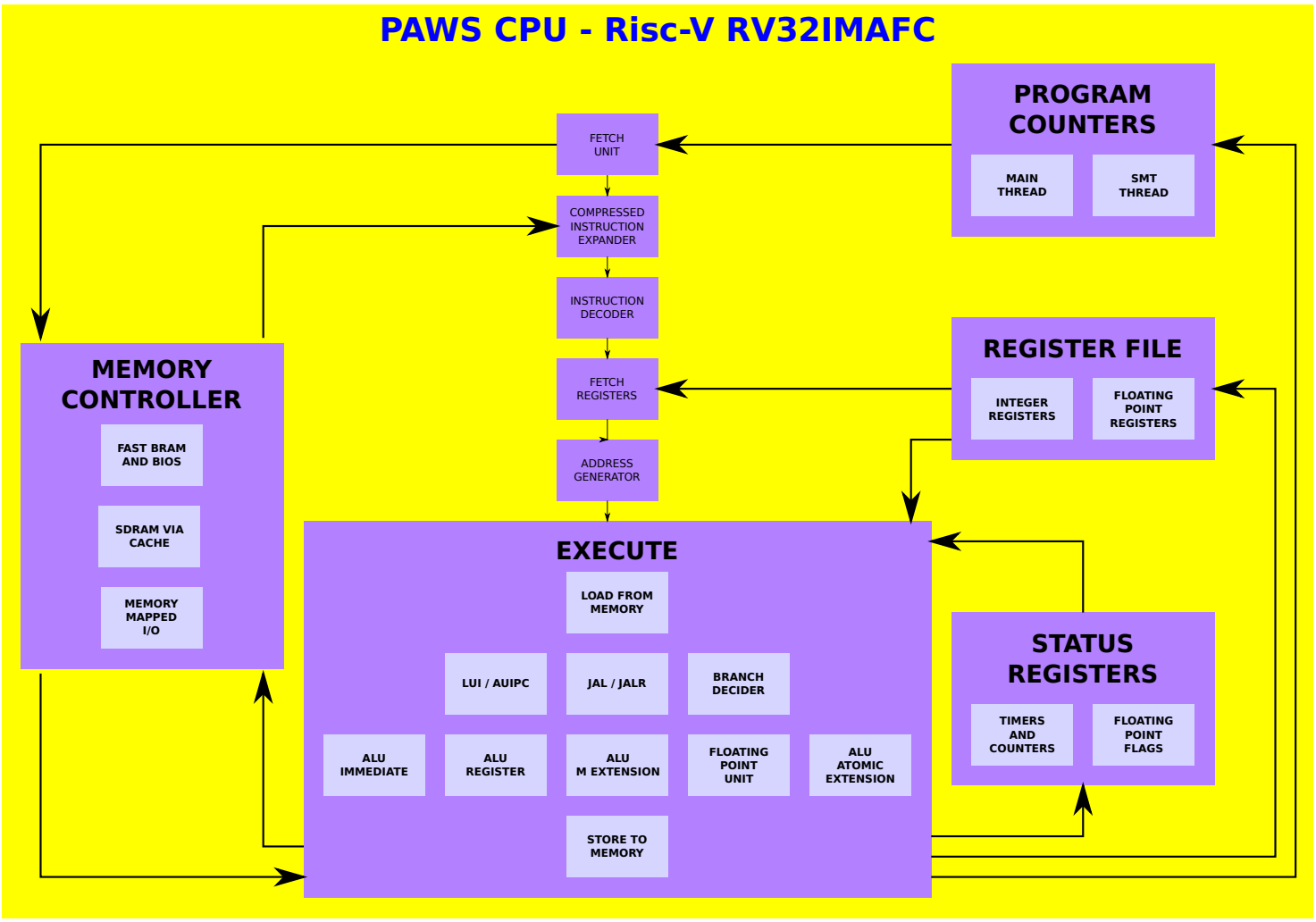
`#include "PAWSlibrary.h"`
 	
`int main( void ) {`
`  INITIALISEMEMORY();`
 	
`  // CODE GOES HERE`
`}`
 	
`// EXIT WILL RETURN TO BIOS`
	Use libPAWS for definitions and helper functions.
	
	MAIN program entry point
	Setup the memory map
	
	Main loop.

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 RV32IMA[F]C[B] CPU – Programming Guide

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 Memory System

The SDRAM has a 32k 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 STACK, and space at the top the SDRAM for SDCARD buffers.

Space is reserved in the fast BRAM memory for the SMT CPU STACK.

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 – 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.
0x10000000 -0x11ffffff	SDRAM PROGRAM + LOADED DATA MALLOC ALLOCATED MEMORY	Program and data storage. Accessed via a cache. SDCARD buffers and structures are allocated at the top of this address range.

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. Allocates the buffers used for SDCARD access and initialises malloc.
unsigned char *filemalloc(unsigned int size)	Allocates space for a file to be read into memory. Preferred over malloc as over allocates to fit with the SDCARD sectors that will be read.

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

PAWS a Risc-V RV32IMA[F]C[B] CPU – Programming Guide

Instruction Set Listings (Configurable at build time)

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.
B EXTENSION Zba Zbb Zbc Zbe Zbf Zbp Zbs	ANDN ORN XNOR BCOMPRESS BDECOMPRESS BFP CLMUL[[H]R] CLZ CTZ PCNT GREV[I] GORC[I] MIN[U] MAX[U] ORC.B PACK[H][U] ROL ROR[I] CLR[I] INV[I] SET[I] EXT[I] SEXT.B SEXT.H SH[[1][2][3]]ADD SHFL[I] UNSHFL[I] XPERM.B XPERN.H XPERM CRC32 CRC32C CMOV CMIX FSL FSR[I]		OPTIONAL Select at build time.
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		OPTIONAL Select at build time. There is no rounding control.

Using PAWS

The PAWS system starts in the BIOS, which initialises the system, and provides a file explorer for the PAW (compiled programs) files in the ROOT DIRECTORY of PARTITION 0 on a FAT16 formatted SDCARD.

Scroll through the available PAW files using LEFT and RIGHT, and select using FIRE 1 (see next page for details of using the keyboard to control PAWS).

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

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.

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.

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 )`	Changes the display layer order. 0 = default, as above. 1 = bitmap between lower sprites and the tilemaps. 2 = bitmap between the tilemaps and the background. 3 = bitmap between the upper sprites and the character map.

File Management

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

libPAWS variables and functions

`unsigned short sdcard_findfilenumber( unsigned char *filename, unsigned char *ext )`	Returns the number of the file in the root directory on the SDCARD, or 0xffff if the file is not found.
`unsigned int sdcard_findfilesize( unsigned short filenumber )`	Returns the size of the file, in bytes.
`void sdcard_readfile( unsigned short filenumber, unsigned char * copyAddress )`	Reads the file into memory.

Example code for loading a file (GALAXY.JPG) into memory

```
#include "PAWSlibrary.h"

void main( void ) {
    INITIALISEMEMORY();

    unsigned char *galaxyfilebuffer;
    unsigned short filenumber;

    while(1) {
        outputstring( "Finding File GALAXY.JPG");
        filenumber = sdcard_findfilenumber( "GALAXY", "JPG" );
        if( filenumber == 0xffff ) {
            outputstring( "FILE NOT FOUND" );
        } else {
            galaxyfilebuffer = filemalloc( sdcard_findfilesize( filenumber ) );
            sdcard_readfile( filenumber, galaxybuffer );

        (void)inputcharacter();    }
}
```

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. Some suggestions for best practice are listed below:

- Only one thread should access the GPU, tile-maps, character map or audio. If both threads attempt to they could be overwriting the control registers leading to an unknown outcome.
- The sprite layers control registers are memory mapped to allow both threads to control the sprites.

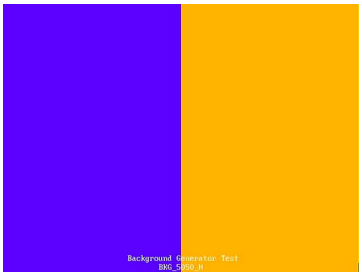
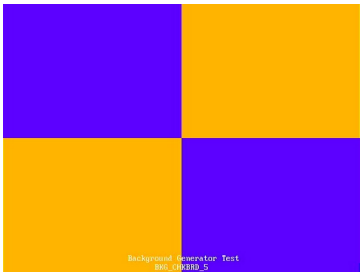
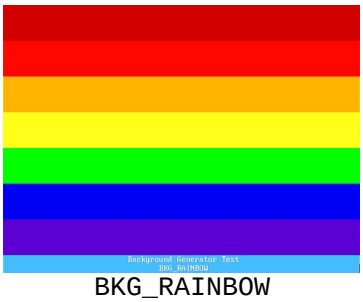
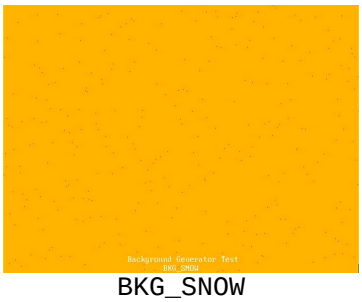
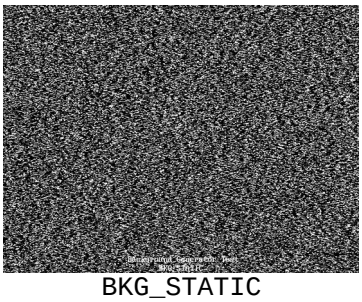
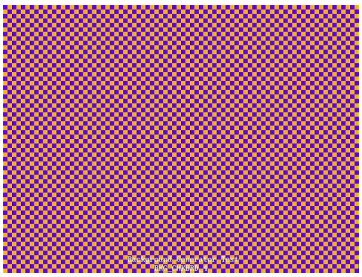
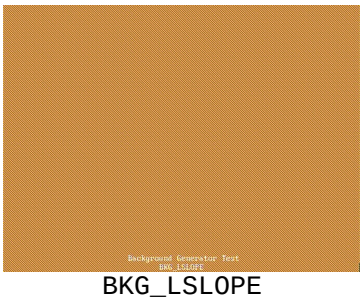
Example code for a simple dual thread program

```
#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.

Background Generator

The background layer shows when there is nothing to display from the layers above. There are 16 named 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.

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

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 );
```

Bitmap and GPU

PAWS displays a 320x240 pixel bitmap with 64 colours, plus TRANSPARENT. To allow for tear-free animation, there are two bitmaps, and it is possible to display one, whilst drawing to the other. The default is to draw and display the same bitmap, numbered 0.

PAWS has a GPU to draw to the bitmap. The functions in libPAWS will setup the GPU, wait for the previous GPU command to complete, and then start the GPU with the required function.

libPAWS variables and functions

`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_cs( void )`	Clears the bitmap to TRANSPARENT and resets the scroll/wrap.
`void gpu_dither( unsigned char mode, unsigned char colour )`	Sets the dither mode and the alternate colour used for filled rectangles, circles and triangles.
`void gpu_pixel( unsigned char colour, short x, short y )`	Plots 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_box( unsigned char colour, short x1, short y1, short x2, short y2 )`	Draws an outline rectangle with corners at (x1,y1) and (x2,y2) in colour. NOTE: Drawn by breaking into 4 lines.
`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. Only pixels in the 45° sectors, numbered 0 to 7 from the top are drawn.
`void gpu_blit( unsigned char colour, short x1, short y1, short tile, unsigned char blit_size )`	Blit a 16x16 tile (32 user definable tiles) to (x1,y1) in colour. Will size to 16x16, 32x32, 64x64 and 128x128.
`void gpu_character_blit( unsigned char colour, short x1, short y1, unsigned char tile, unsigned char blit_size )`	Blit an 8x8 character (256 user definable characters, defaults to the IBM character set) to (x1,y1) in colour. Will size to 8x8, 16x16, 32x32, 64x64.
`void gpu_colourblit( short x1, short y1, short tile, unsigned char blit_size )`	Blit a 16x16 tile (32 user definable tiles) to (x1,y1). Will size to 16x16, 32x32, 64x64 and 128x128.
`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. NOTE: Vertices should be presented clockwise from the top.
`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 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 clockwise from the top.
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).
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.
void gpu_printf(unsigned char colour, short x, short y, unsigned char size, 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. 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, 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. NOTE: Escape characters are not processed, the corresponding character code is output as a character.
void set_blitter_bitmap(unsigned char tile, unsigned short *bitmap)	Define one of the 32 16x16 blitter tiles.
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 32 16x16 colour blitter tiles.
void draw_vector_block(unsigned char block, unsigned char colour, short xc, short yc)	Starts the drawing of one of the 32 user definable vector (line drawn) objects, centred at (x,y) in colour.
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.
void bitmap_scrollwrap(unsigned char action)	action == 1 LEFT, == 2 UP, == 3 RIGHT, == 4 DOWN, == 5 RESET

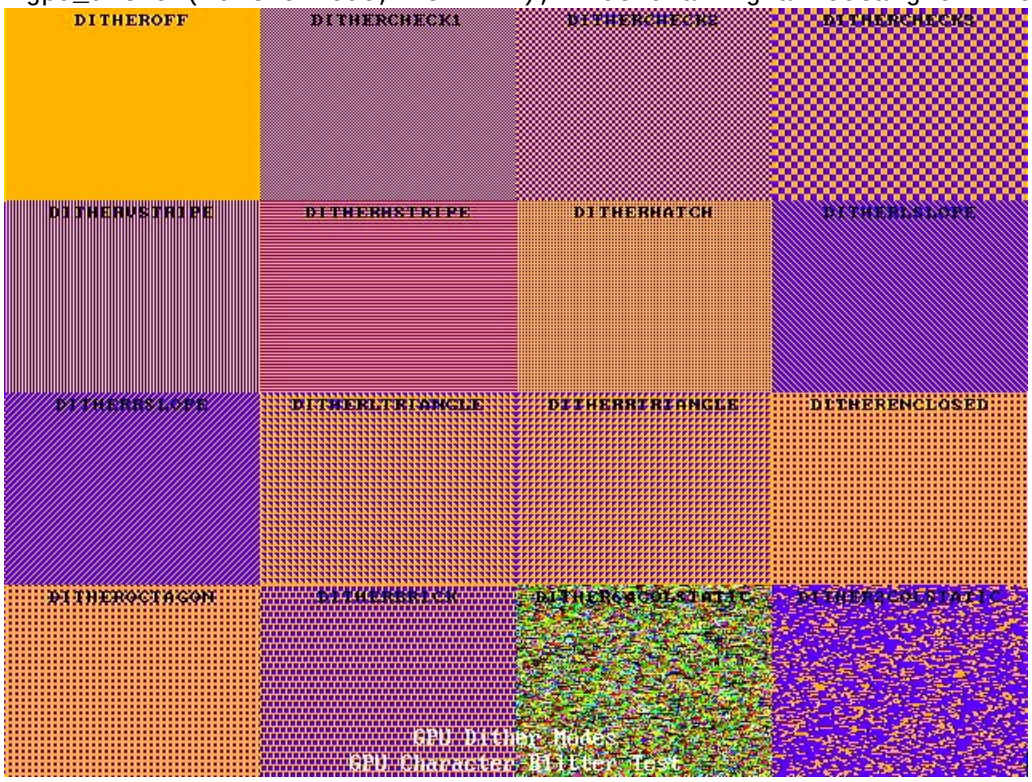
Example code for a tear-free animation

```
#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 Modes

When drawing rectangles, filled circles or filled triangles, the GPU can apply one of 16 “dither” patterns.

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



GPU Blitters

PAWS has 3 blitters:

- A single colour 16x16 tile blitter
  - TRANSPARENT pixels are ignored when drawing to the bitmap
- A single colour 8x8 tile blitter
  - Defaults to the IBM character set, used to draw text to the bitmap
- A colour 16x16 tile blitter
  - TRANSPARENT pixels are ignored when drawing to the bitmap

GPU Vector Blocks