

PAWS a Risc-V RV32IMACB CPU – Programming Guide

About

PAWS is a project to occupy my time in retirement, 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.

It is a development from the work I did on the J1 CPU, which is a 16-bit CPU designed to run Forth natively, to which I added a display via HDMI, plus various input/out facilities such as a UART, button input, LED output and basic audio.

The J1 CPU has been swapped out for a Risc-V RV32 processor, as this can easily be programmed via C using GCC, in addition to being relatively straightforward to being implemented on an FPGA.

A support library, libPAWS, for easy access to the hardware is provided. 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>).

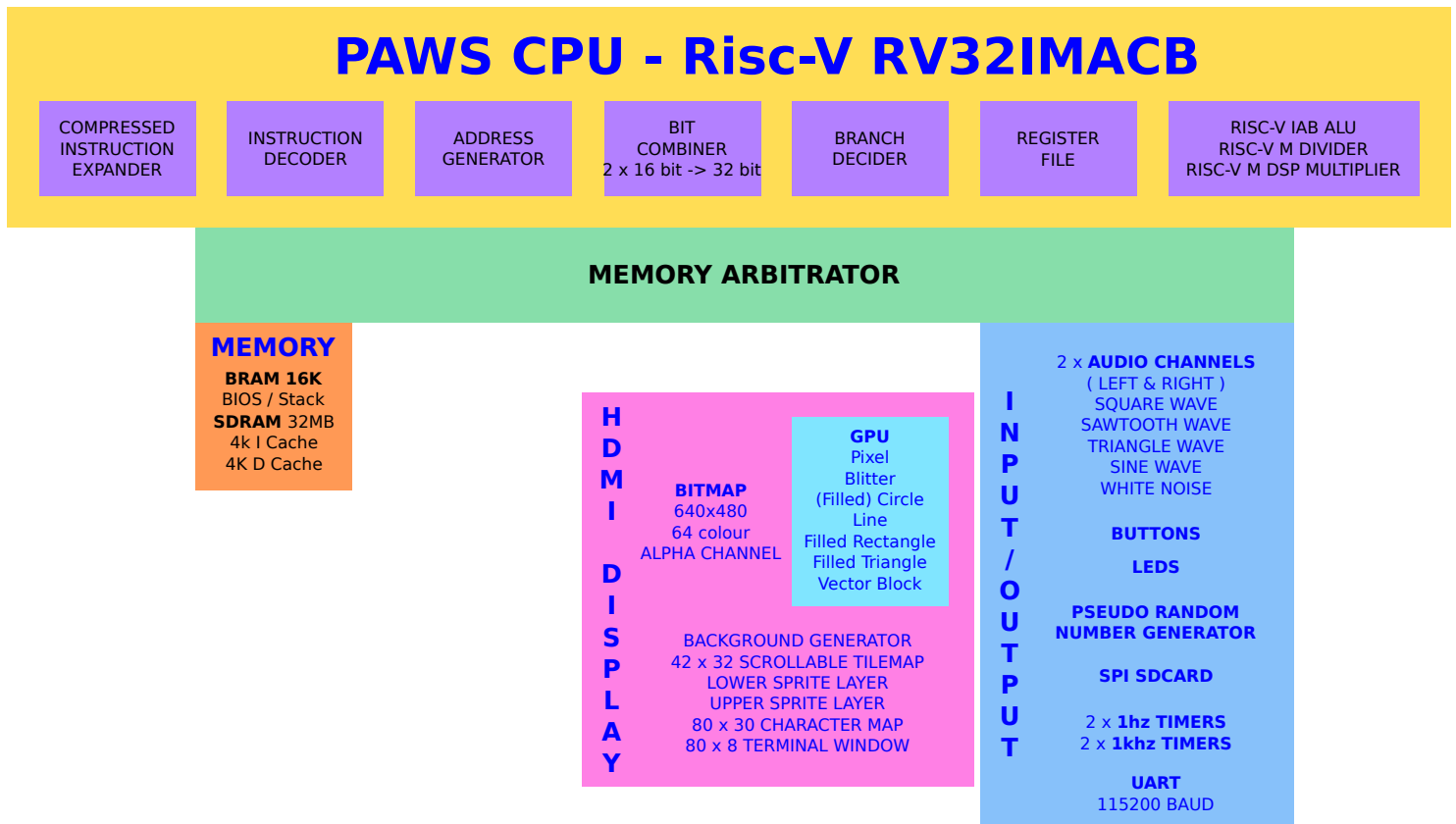
My coding style may not result in the *best* design, the aim was to create a design that could be easily understood.

For example, in the Risc-V Bit Manipulation Extension there is an instruction SBSET that sets a bit to 1. On the left is the reference implementation, using an OR and a SHIFT. On the right is the chosen code, which I believe to be easier to understand, which explicitly shows the chosen bit being set.

<pre>circuitry SBSET(input sourceReg1, input shiftcount, output result) { result = sourceReg1 (1 << shiftcount); }</pre>	<pre>circuitry SBSET(input sourceReg1, input shiftcount, output result) { result = sourceReg1; ++: result[shiftcount, 1] = 1b1; }</pre>
--	--

PAWS a Risc-V RV32IMACB CPU – Programming Guide

PAWS CPU and SOC



The PAWS CPU is a Risc-V RV32IMACB that implements only the features needed to run GCC compiled code.

- No interrupts.
- Machine mode only.

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.

The bit manipulation extension was implemented to increase code density, as when output by GCC they will replace multiple RV32I instructions, again leading to quicker code execution due to reduced memory access.

PAWS BIOS



Upon startup PAWS boots to the BIOS, which initialises the display, clears any input/output buffers, and reads the ROOT DIRECTORY from PARTITION 0 of a FAT16 formatted SDCARD.

PAW (compiled programs) files are display for selection, scrolling through the available files using “FIRE 2” and selecting a file for loading and executing using “FIRE 1”.

Upon selection, the BIOS will load the selected PAW into SDRAM, reset the display, and launch the selected PAW program.

PAWS a Risc-V RV32IMACB CPU – Programming Guide

Compiling Programs For PAWS

The default language for PAWS is C, specifically GCC.

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
<pre>#include "PAWSlibrary.h" void main(void) { INITIALISEMEMORY(); while(1) { } }</pre>	<p>Use libPAWS for definitions and helper functions.</p> <p>MAIN program loop Setup the memory map</p> <p>Main loop.</p>

Compile your code using the helper shell script. For example, to compile the included asteroids style arcade game, `./compile_SDRAM.sh c/asteroids.c`. This will compile the program to `build/code.PAW`, which can be copied to the SDCARD for loading via the BIOS.

PAWS a Risc-V RV32IMACB CPU – Programming Guide

Colours

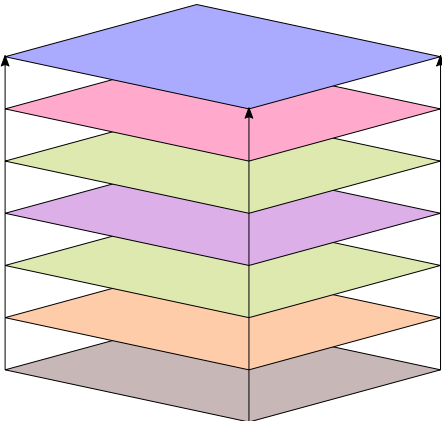
PAWS uses a 6-bit colour attribute, given as RRGGBB. This gives 64 colours, specified in decimal as per the table below. Names defined in libPAWS are given below the decimal representation.

0	1	2	3	4	5	6	7
BLACK		DKBLUE	BLUE				
8	9	10	11	12	13	14	15
DKGREEN			DKCYAN	GREEN			CYAN
16	17	18	19	20	21	22	23
			PURPLE		GREY1		
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
DKRED		DKMAGENTA					
40	41	42	43	44	45	46	47
DKYELLOW		GREY2					
48	49	50	51	52	53	54	55
RED			MAGENTA				
56	57	58	59	60	61	62	63
ORANGE				YELLOW			WHITE

In addition, for layers that allow, tilemap, bitmap and character map, there is a TRANSPARENT attribute, which allows the layer below to show through. See relevant sections of the documentation for further explanation.

PAWS a Risc-V RV32IMACB CPU – Programming Guide

Display Structure

<div>Default Display Layer Ordering</div> <div><div>Terminal Window Character Map Upper Sprites Bitmap Lower Sprites Tilemap Background</div></div> <div><p>The display, 640 x 480 pixels in 64 colours, is output via HDMI.</p><p>It is formed of layers, starting with the terminal window at the top, down to the Background.</p><p>The terminal window can be switched off, otherwise it displays as an 80 x 8 character window at the bottom of the display.</p><p>All other layers, have transparency so that if no pixel is showing in that layer, the layer below shows.</p></div>

libPAWS variables and functions

<code>void await_vblank(void)</code>	Waits for the screen vertical black to start.
<code>void screen_mode(unsigned char screenmode)</code>	<p>Changes the display layer order.</p> <p>0 = default, as above. 1 = bitmap between lower sprites and the tilemap. 2 = bitmap between the tilemap and the background.</p>

PAWS a Risc-V RV32IMACB CPU – Programming Guide

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.

Address Range	Memory Type	Usage
0x00000000 – 0x00004000	Fast BRAM	0x00000000 – 0x00001388 BIOS 0x00004000 – 0x00002000 STACK
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.
0x1000000 -0x1ffffff	SDRAM	Program and data storage. Accessed via instruction and data caches. SDCARD buffers and structures are allocated at the top of this address range.

libPAWS variables and functions

<code>unsigned char *MEMORYTOP</code>	Points to the top of unallocated memory.
<code>void INITIALISEMEMORY(void)</code>	Sets up the memory map using parameters passed from the BIOS. Allocates the buffers used for SDCARD access, and correctly sets MEMORYTOP.
<code>unsigned char *memoryspace(unsigned int size)</code>	Returns a pointer to a buffer of at least size bytes, allocated from the top of the free memory space. Aligned to 16-bit address.
<code>unsigned char *filememoryspace(unsigned int size)</code>	Returns a pointer to buffer of at least size bytes, allocated from the top of the free memory space. Aligned to 16-bit address. This should be used for preference when allocating memory in which to read a file, as the buffer will contain sufficient space to account for the minimum block size of the SDCARD.

NOTE: memoryspace and filememoryspace are similar to the standard C function malloc. There is no mechanism for freeing memory in libPAWS.

PAWS a Risc-V RV32IMACB CPU – Programming Guide

File Management

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

libPAWS variables and functions

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

NOTE: memspace and filememspace are similar to the standard C function malloc. There is no mechanism for freeing memory in libPAWS.

Example code for oading a file (GALAXY.JPG) into memory
<pre>#include "PAWSlibrary.h" void main(void) { INITIALISEMEMORY(); unsigned char *galaxyfilebuffer; unsigned short filenumber; while(1) { outputstring("Finding File GALAXY.JPG"); filenumber = findfilenumber("GALAXY", "JPG"); if(filenumber == 0xffff) { outputstring("FILE NOT FOUND"); } else { galaxyfilebuffer = filememspace(findfilesize(filenumber)); sdcard_readfile(filenumber, galaxybuffer); (void)inputcharacter(); } } }</pre>