sc68 technical features

Motorola[tm] 68000 processor full emulator [Atari ST[tm] and Amiga[tm] CPU]
Yamaha[tm] 2149 full emulator [Atari ST[tm] sound chip]
MFP 68901 emulator (Atari ST[tm] timers only)
MicroWire([tm]) emulator (Atari ST[tm]-STE specific)
Paula (Amiga[tm] sound chipset)

Motorola[tm] 68000 microprocessor [Mc 68000]

68000 processor (aka Mc68000 or 68K) was the first of the 32 bit processor family. It appears on personnal computers such as *Apple Machintosh[tm]*, *Sinclair QL[tm]* and later *Atari ST[tm]*, and *Commodore Amiga[tm]*.

Mc68000 is a 64 pin plastic package, with separate data and address bus and a very complex set of signal.

A large amount of register and a very complete instruction set combined with 14 addressing modes and 3 operand sizes (8, 16 and 32 bit) made it the very most attractif microprocessor in its time. According to most programmers Mc68000 still be one of the "funniest" microprocessor to program.

Mc68000 registers

Data registers	D0-D7
Gerenal purpose address register	A0-A6
Stack pointer	A7 or SP
User mode stack pointer register	USP
Status Register (including CCR)	SR
Code Condition Register (part of SR)	CCR

Mc68000 addresing modes

Short absolute	Long absolute
Data register direct	Address register direct
Status register direct	Immediat
Quick (short) immediat	Indirect
Postincrental indirect	Predecrental indirect
Indirect with displacement	Indexed indirect with displacement
PC relatif	Indexed PC relatif with displacement

Mc 68000 emulation

Overview

The Mc68000 emulator was designed in order to execute 68K music player, which is a main reason of its being. Doing a 68K emulator is not a very easy things and it will be great if this one could be used for other tasks.
The application need maximum control over emulation behaviour such as register access, memory access, IO-processor hot plugin...
The emulation must be fast enought to avoid processor overload.

Features

100% optimized C code

For optimization purposes, the emulator use quiet large function tables. A single sourced program has been developed to create the functions called by these tables. The functions are stored into 16 files called line0 line1 ... lineF. Functions's body is composed of preprocessor macros to improve code generation and maintenance. The whole makes the

emulator library quiet large but it seems to be the only acceptable way to design a decent Mc68000 emulator.

99% portable
: The project portabilty has not currently been tested, but it is expected to work fine with all 32 bit processors of any endianess. 64 bit processors may uncounter some expected errors `:-(`

Very flexible IO processor plugin
: IO processors are managed separatly and could be hot plugged on 68K IO adress space at anytime. Praticaly, IO processors are pluged in at initialization time according to wanted hardware (Atari ST[tm] or Amiga[tm]).

Cycle-precision very fast IO emulation
: IO-processors emulation is design to permit a high cycle precision emulation without overloading processor. Classical emulator execute IO emulation at each instruction emulation which really increases emulation processing time. To avoid this some emulator emulate IO processor after a given number of cycle (or instruction) but IO emulation lost cycle precision. Atari ST[tm] soundchip technics to improve YM-2149 sound render have been developed over years really need maximum precision to produce a good emulation. For this reason, a buffered technics has been design that allow each chip to "remember" write access and to perform it when really needed. By another mechanism, the 68K emulator always knows when an IO processor will interrupt and which one. The emulation inner loop only need to test if it is time to interrupt and to call the corrsponding interrupt procedure.

Debug mode
: A debug mode of the emulator library could be compile by setting suitable preprocessor define. In this mode memory access are done by byte per byte. For each byte of 68K memory a status byte including an *hardware breakpoint* and some access flags (*read/write/execute*). These flags are used by sc68 utilities to calculate music time. Since musics are programs, calculating a music time is equivalent to know when a programm finish. This is known as a very complex problem to solve. A specific use of this memory access flags allows sc68 time calculator to success in this task in more than 99% of cases, and by the way to save many very boring time of listening musics a clock in the hand `:-)`

Limitations

Supervisor
: Most (if not all) music player on Atari ST[tm] and Amiga[tm] directly address the sound hardware. The mapped address of such chip is in privilegied address space, which means that the 68K must be in supervisor mode to access this address space. The emulator assumes that the processor is always in supervisor mode to avoid useless test that would increase processor load.

BCD instructions
: For lazyness reason the BCD instructions (*binary coded decimal*) emulation has been omitted. It will not be a great deal to add them if neccessary, but it is uncommon to encounter this kind of instruction in a music player.

Yamaha[tm] YM-2149 [Software-controlled Sound Generator]

Overview **[ref: YM2149 catalog No: LSI-2121492]**

The SSG is an NMOS-LSI device designed to be capable of music generation. It only requires the microprocessor [68000] to iniatilize its regiser array, thus reducing the load on the CPU. Music generation is carried out by three sequence square wave generator, noise generator, and envelop generator accordig to the set of parameters. This allow for the generation of music, special effects, warnings and various other types of sound.

Features

- 5V single power supply
- Easy connection to 8 bit or 16 bit CPU
- Simple connection to external system through 2 sequence 8 bit I/O port
- Wide voicing range of 8 octaves
- Smooth attenuation by 5 bit envelope generaror
- Built-in 5 bit D/A convertor
- Input of double frequency clock can be handled by built-in clock frequency divider
- TTL compatible level
- 40 pin plastic package
- Pin compatible with AY-3-8910 manufactured by GI

Description of functions

All functons of the SSG are controled by the 16 internal registers. The CPU need only write data to the internal registers of the SSG. The SSG itself generates the sound.
Sound is generated by the following blocks:

YM2149 function blocks

Music generator	Square waves generator for each channel
Noise generator	Pseudo-random waveforms generator for each channel
Mixer	Music and noise output are mixed for each channel
Level control	Constant or variable level is given for each channel
Envelop generator	Generate various type of level envelop (only one)
D-A convertor	Sound is output at the level determined by the level control for each channel

The CPU can read the contents of the internal registers with no effect on sound.

Register Array

Register Array table

		B7	B6	B5	B4	B3	B2	B1	B0
R0	Tone period [channel A]	8 bit fine tone adjustement							
R1						4 bit rought tone adjustement			
R2	Tone period [channel B]	8 bit fine tone adjustement							
R3						4 bit rought tone adjustement			
R4	Tone period [channel C]	8 bit fine tone adjustement							
R5						4 bit rought tone adjustement			
R6	Noise period				5 bit noise period				
R7	I/O port and mixer settings	I/O		Noise			Tone		
		IOB	IOA	C	B	A	C	B	A
R8	Level [channel A]				M	4 bit level			
R9	Level [channel B]				M	4 bit level			
RA	Level [channel C]				M	4 bit level			
RB	Envelope period	8 bit fine adjustement							
RC		8 bit rough adjustement							
RD	Shape of envelope					CONT	ATT	ALT	HOLD
		_				0	0		
		/_				0	1		
		\/\				1	0	0	0
		_				1	0	0	1
		\/				1	0	1	0
		\‾				1	0	1	1
		/\/\				1	1	0	0
		/‾				1	1	0	1
		/\				1	1	1	0
		/_				1	1	1	1
RE	I/O port A data	8 bit data							
RF	I/O port B data	8 bit data							

Setting of music frequencies [R0~R5]

The frequencies of the square wave generated by the music generators for the three channels (A,B and C) are controlled by registers R0 through R5. The oscillation frequency Ftone is obtained in the following manner from the value of the register TP.

$$Ftone = Fmaster / 16 * TP$$

Where:

`Fmaster`

Master clock frequency (2Mhz for the Atari ST^{tm}).

TP

Tone period given by registers R0~R5 (`TPA=R0+256*R1...`)

Setting of noise generator [R6]

The Random Number Generator of the 8910 is a 17-bit register. The input to the register is bit0 XOR bit2 (bit0 is the output).
The noise frequency Fnoise is obtained from the register NP in the following manner:

$$Fnoise = Fmaster / 16 * NP$$

Where:

NP

Noise period given by the 5 less signifiant bit of register R6.

Setting of mixer and I/O ports [R7]

The mixer is used to combined music and noise components. The combination is determined by bits B5~B0 of register R7. Both Noise (B5~B3) and Tone (B2~B0) bit are active when a "0" is written.

The I/O port settings have nothing to do with sound generation. Selection of input/output for the I/O ports is determined by B7 and B6 of register R7 for respectively port-B and port-A. Input is selected when "0" is written.

Level control [R8~RA]

The audio level output from the D/A convertors for the three channels (A,B, and C) is respectively adjusted by the registers R8, R9 and RA. Bit B4 defines the mode. When mode is "0" the level is determined by the bits B3~B0. When mode is "1" the level is determined by the 5 bit output of the envelop generator (E4~E0), B3~B0 are ignored.

Setting of envelope frequency

The envelope update frequency FEA is obtained as follow from the envelope setting period value EP.

$$FEA = Fmaster / 8 * EP$$

Where:

EP

Envelope period given the registers RC~RB.

The envelope repetition frequency FE is obtained as follow from the envelope update frequency FEA

$$FE = FEA / 32$$

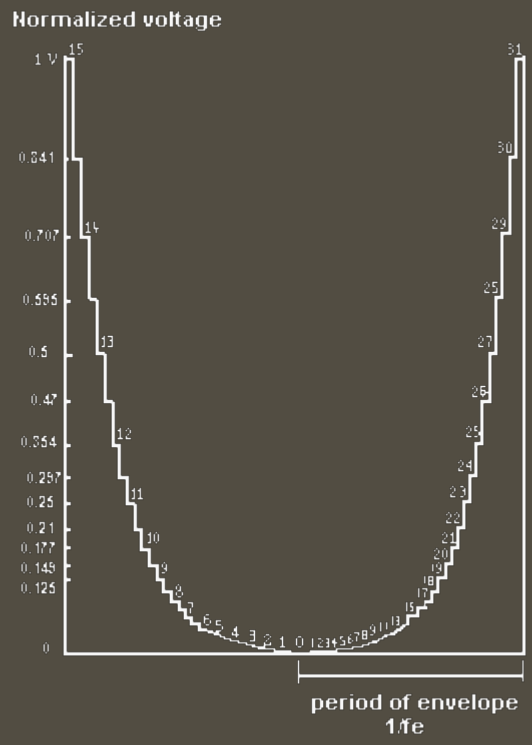
Envelope shape control [RD]

The envelope generator counts the envelope clock FEA 32 times for each envelope pattern cycle. The level is determined by the 5 bit output (E4~E0) of the counter. The shape of this envelope is created by increasing, decreasing, stopping or repeating this counter. The shape is controlled by bits B3~B0 of the register RD (see *Register Array* table). Envelope shape is restarted each time the register RD is written.

I/O port data hold [RE and RF]

Register RE and RF are used to store the data written from the CPU to the I/O ports A and B respectively.

D/A convertor



Each of the three channels is equiped with a D/A convertor which converts the calculated digital values to analog signals for output.

When the maximum amplitude is normalized to 1V, the levels shown in figure below are obtained. This convertion from a linear input to a logarithmic output provides a wide dynamic range and a natural feeling attenuation.