Devpac Falcon Library

Johan Karlsson
email: d92jk@efd.lth.se

Version 5.2
16 January 1995

Contents

1 Introduction **1**
1.1 Easy programming 1
1.2 How to use it 1

2 Routine descriptions **2**
2.1 Cache 2
2.2 Cls 2
2.3 Convpal1 2
2.4 Cookie 2
2.5 Dspmod 2
2.6 Exit 2
2.7 Fade 2
2.8 Fade2 2
2.9 Frame 2
2.10 Gem 2
2.11 Getpar 3
2.12 Goraud 3
2.13 Hline1 3
2.14 Ice 3
2.15 Loadfile 3
2.16 Releasem 3
2.17 Savefile 4
2.18 Setfv 4
2.19 Setvideo 4
2.20 Shrtones 4
2.21 Timer 4
2.22 Trig 5
2.23 Window 5
2.24 Winevent 5

3 History **6**
3.1 Version 5.2 6
3.2 Version 5.1 6
3.3 Version 5 6
3.4 Version 4 6
3.5 Version 3 6
3.6 Version 2 6
3.7 Version 1 6

4 Credits **6**
4.1 Dspmod 6
4.2 Goraud 6
4.3 Cache 6
4.4 Fade 6

5 Address **6**

1 Introduction

1.1 Easy programming

This is an assembler library for the Atari Falcon. By using this I hope to make programming a bit faster and easier. It includes a lot of subroutines which most assembler programmers use quite frequently as well as some less frequently used graphic routines. I have tested all routines and they seem to work ok, but I can't guarantee it. For everything to work properly use Devpac 3 with ignore multiple includes active. Most things will probably work nice with Devpac 2 too.

If you have made a routine I would appreciate if you sent it to me, that way I could include it in the Falcon Library for the benefit of everyone. Please send any questions, bugreports or contributions to me. My address is at the end of this document.

1.2 How to use it

Following are short descriptions of all included routines. Most are subroutines and should be called with 'bsr' or 'jsr' unless the word macro follows the routine name in which case it is a macro. If a routine says 'destroys a lot' then you should save all the registers you are using except a7 which is never ruined. Some routine descriptions even contains an example, so hopefully there will be no problems understanding them.

2 Routine descriptions

2.1 Cache

Cache equates. You'll need devpac 3 for this one I think. Here they are:
ENABLE_CACHE *equ* 1
FREEZE_CACHE *equ* 2
CLEAR_INST_CACHE_ENTRY *equ* 4
CLEAR_INST_CACHE *equ* 8
INST_BURST_ENABLE *equ* 16
ENABLE_DATA_CACHE *equ* 256
FREEZE_DATA_CACHE *equ* 512
CLEAR_DATA_CACHE_ENTRY *equ* 1024
CLEAR_DATA_CACHE *equ* 2048
DATA_BURST_ENABLE *equ* 4096
WRITE_ALLOCATE *equ* 8192

```
ex move.l #ENABLE_CACHE+ ENABLE_DATA_CACHE+
WRITE_ALLOCATE,d0
movec d0,CACR ; turns on the caches
```

2.2 Cls

@cls???
clears ??? bytes.
In a0.l=start address
(destroys a lot)

2.3 Convpal1

@convert_stpal
Convert an old st colour value to a new falcon one
In d0.w=old st colour
Out d0.l=falcon colour
(destroys d1-d3)

2.4 Cookie

@getcookie
Finds and returns a Cookie. Supervisor only!
In d1.l=Cookie name
Out d0.l=Cookie value or -1
(destroys a0)

2.5 Dspmod

@dsp_play
Sets interupts and plays some soundtracker music. Modules mustn't have more than 4 voices. Supervisor only.
In a0.l=Module adr.
(destroys a lot)

@dsp_stop
Stops playing the music and restores interupts. Supervisor only.
(destroys a lot)

2.6 Exit

@exitifspace *macro*
branches to exit if space has been pressed. Supervisor only!
(destroys nothing)

2.7 Fade

@fadetrupix
Fades a true colour pixels from one colour to another. It will fade 1/31 part each time you call this rout.
In a0.l=pointer to colour to change
a1.l=pointer to target colour
(destroys d1-d4; adds 2 to a0-a1)

2.8 Fade2

@fadetrupix2
Fades a true colour pixels from one colour to another. It will fade 1/31 part each time you call this rout. The difference between this one and FADE.S is that this one fades two pixels at once and is slightly faster.
In a0.l=pointer to colours to change
a1.l=pointer to target colours
(destroys d1-d4; adds 4 to a0-a1)

2.9 Frame

This is a program frame that may be used as a start for new programs. Just load it and fill in the blanks (there is a lot of them).

2.10 Gem

@gemdos *macro*
Calls trap #1
In #functionnbr, #stackcorrection

```
ex @gemdos 1,2 ; waits for a key
```

@xbios *macro*
Calls trap #14

In #functionnbr, #stackcorrection

```
ex @xbios 37,2 ; waits for vsync
```

@gem *macro*
Calls a trap
In #functionnbr, #trapnbr, #stackcorrection

```
ex @gem 1,1,2 ; waits for a key
```

2.11 GETPAR

(include right after releasem.s)

Gets all parameters sent to the program. Must be directly after releasem.
Out a0.l=Address where the parameters are kept

```
ex include releasem.s ; release unused memory,
this is line nr 1 of the program
include getpar.s ; get adr.  where the
parameters are kept
move.l a0,paradr ; save the result

paradr ds.l 1
```

2.12 GORAUD

@goraud
Draw a goraud shaded polygon in true colour. There's no clipping so don't draw too large polygons. Call @initgoraud once before using this one!

In a1.l=screen address
a2.l=adr to colour table
*new_crds=following table: [x,y,i]*nbr_of_points*
d6.w=nbr. of points
xres equ horizontal resolution
(destroys a lot)

```
ex xres equ 384 ; overscaned lowres screen
bsr @initgoraud ; call once only
move.l scradr,a1 ; screenaddress
move.l #g_red,a2 ; I want a red poly
moveq #4,d6 ; four points
bsr @goraud ; draw it
new_crds dc.w 10,10,0 ; a dark point
dc.w 200,20,$7fff ; a bright point
dc.w 140,140,$7fff/2
dc.w 25,50,$7fff/4
```

The colourtable consists of 32 words, each a true colour value. word nr 0 is the darkest and word nr 31 the brightest. This is followed by 16 empty words. Supplied colours: g_lightred, g_green, g_brown, g_white new_crds is a table that consist of a pair of coordinates followed by a brightnes factor. The brightnes factor is a number between 0 and #$7fff.

@initgoraud
This subroutine must be called once before using @goraud.
(destroys a lot)

2.13 HLINE1

@drawhline1
Draws a horizontal line in 8 bitplane mode. Clipping is implemented. This routine only uses the six first bitplanes.
In a0.l=screenadr
d0=xmin
d1=xmax
d2.l–d4.l =6 bitplanes
xres equ horizontal screen resolution
(destroys d0–d7/a0–a1)

```
ex move.l screen+xres*50,a0 ; ypos=50
move #-34,d0 left xcord
move #67,d1 right xcord
move.l #$0,d2 don't set bitplane 0 or 1
move.l #$0000ffff,d3 set bitplane 3
move.l #$ffffffff,d4 set bitplane 4 and 5
bsr @drawhline1
```

2.14 ICE

@icedecrunch
Checks if the data is icepacked and possibly unpacks it.
In a0.l=Data adr.
(destroys nothing)

2.15 LOADFILE

@loadfile
Loads a file and possibly unpacks it (ice 2.40).
In a5.l=adr. to a filename terminated by zero
a6.l=dest.
d7.l=filelength
(destroys a lot)

2.16 RELEASEM

This routine releases all memory not needed by the program. Include this rout at the top of your program and the memory will be relesed properly.

```
ex include releasem.s ; release unused memory,
```

```
this is line nr 1 of the program
```

2.17 SAVEFILE

@savefile
Saves data as a file.
In a5.l=adr. till nollavslutat filnamn
a6.l=source
d7.l=fillängd
(destroys a lot)

2.18 SETFV

@setfv
Sets the falcon video registers. The data that is written to the video registers must be a .FV (Falcon Video) file. Supervisor only. The advantages of using this instead of **@setvideo** is that this one will work with screen enhancers like screenblaster, **@setvideo** won't and by using .FV files it's also possible to get higher resolutions.
In a0.l=adr. to Falcon Video data
Out d0.l: 0=no error -1=error, no .FV data
(destroys d0–d2/a0–a2)

@savefv
Saves the falcon video registers to memory. Supervisor only.
(destroys a0–a1)

@resorefv
Restores the saved falcon video registers. Supervisor only.
(destroys d0–d2/a0–a2)

2.19 SETVIDEO

@setvideo
Save and set resolution.
In d7.w=mode
modeequates: vertflag, stmodes, overscan, pal, vga, col80, bps1–16
(xbios, destroys d0–d2/a0–a2)

```
ex move #pal+bps16,d7
bsr @setvideo
```

@restorevideo
Restores the saved resolution.
(xbios, destroys d0–d2/a0–a2)

@setvadr
Sets the physical and logical screenaddress.
In d0.l=screenadr.
(xbios, destroys d0–d2/a0–a2)

@savevadr
Saves the current screenadr.
(xbios, destroys d0–d2/a0–a2)

@restorevadr
Restores the saved screenadr.
(xbios, destroys d0–d2/a0–a2)

2.20 SHRTONES

@super
enter supervisor mode
(gemdos $20, destroys d0-d1/a0-a1)

@user
returns to user mode
(gemdos $20, destroys d0-d1/a0-a1)

@waitvbl
waits for a vertical blank
(xbios #37, destroys d0-d2/a0-a2)

@waitkey
waits for a keypress, no echo
Out d0.b=ascii value
(gemdos #7, destroys a0)

@exitifkey
quits if a key has been pressed
(gemdos #11,#7,#0, destroys a0)

@quit
terminate process
(gemdos #0)

@print
prints a string on the screen.
In a0.l=address to a null terminated ascii string
(gemdos #9, destroys a0)

2.21 TIMER

@colour *macro*
sets backround colour to value. Supervisor only!
In value.l
(destroys nothing)

2.22 TRIG

All results are taken from a 1440 entries large lookup table.

@sin
Returns the sine of an angle.
In d0.w=angle in degrees with six decimal bits (the legal values are 0-$59ff or 0-359.999 degrees)
Out d1.w=sin(angle), a word with one sign bit and 15 decimal bits.
(destroys d0/a0)

@cos
Returns the cosine of an angle.
In d0.w=angle in degrees with six decimal bits (the legal values are 0-$59ff or 0-359.999 degrees)
Out d1.w=sin(angle), a word with one sign bit and 15 decimal bits.
(destroys d0/a0)

```
ex move #180*64,d0 ; make it 180 degrees
bsr @sin ; d1 will now contain 0
move #180*64,d0 ; d0 is destroyed so we had
better enter the value again
bsr @cos ; d1 will now contain -1 ($8000)
```

2.23 WINDOW

(VDILIB and AESLIB must be included at the end of the program)

@createwindow
Creates and opens a simple gem window. The size (xwidth,ywidth) is the workarea of the window.
In wtype equ %info move full close name xstart.w, ystart.w ,xwidth.w ,ywidth.w ,windowname (string terminated by 0)
Out w_handle.w, ap_id.w, screenxmax.w ,screenymax.w
(destroys a lot)

@recalcwindow
Recalculates the window size.
(destroys a lot)

@moveit
Moves the window. May be called at every vm_moved(=28) event.
In a0.l=adr to messagebuffer
(destroys a lot)

@drawrsrc
Draws the rsrc. This function doesn't care about clipping, so you should probably not use it.
In a0.l=adr to rsrc
(destroys a lot)

@updatersrc
Draws the rsrc. Use this when receiving update events(=20). This function takes care of all clipping.
In a0.l=adr to rsrc
(destroys a lot)

@topwindow
Activates the window. May be called at every vm_topped(=21) event.
In a0.l=adr to messagebuffer
(destroys a lot)

@bottomwindow
Bottoms the window if it's mine. May be called at every vm_bottomed(=33) event.
In a0.l=adr to messagebuffer
(destroys a lot)

@button
Returns the object number that was clicked on. This function may be called at every mousebutton event.
In a0.l=adr. to rsrc (It automatically takes the x and y coordinates from int_out)
Out d0.w=object that was pressed or -1.
(destroys a lot)

@loadrsrc
Loads a resource file and creates a window containg the object in the file. You don't have to call @createwindow if you use @loadrsrc.
In wtype equ %info move full close name windowname (string terminated by 0)
a0.l=address to a nul terminated filename
Out a0.l=address to the resource data
w_handle.w, ap_id.w, screenxmax.w, screenymax.w
(destroys a lot)

2.24 WINEVENT

@dowindowevents
For the lazy one. This function waits for events and takes care of everything but the close button or if someone clicked on an object. If one of these events occur the program branches to closeevent or buttonevent. Those subroutines you have to make yourself.
In a0.l=adr. to rsrc
w_handle, ap_id (These are created automatically

if you use @createwindow or @loadrsrc)
Out buttonevent: d0.w=number of the object pressed
closeevent: nothing
(destroys a lot)

```
ex ;create a window;
move.l rsrcadr,a0
bra @dowindowevents ; it will never return so
you may use 'bra' or 'jmp'

buttonevent ; d0.w will contain the
object that was pressed
cmp #1,d0 ; was it my button?
beq mybutton ; yes!
rts ; no!

closeevent
bsr @exitwindow ; close window
(rsrc_free) ; only if you have used @loadrsrc
bra @quit ; quit
```

3 History

3.1 Version 5.2

GFA Basic source code for using FV files are now included. 320*240 rgb FV files added.

3.2 Version 5.1

There was an error in the FV Editor documentation that is removed now. The FV editor has also been somewhat improved.

3.3 Version 5

A bug in the FV routines has been removed (I had missed some important registers). You cannot use old FV files with the new routines! A FV editor is now supplied to test or edit FV files and there is also some documentation describing the FV file format. There has also been some corrections in the main manual (the one you are reading now) and a postscript version has been added.

3.4 Version 4

This LaTeX document is now included. True colour fade rout optimised (about 20% faster) and a new two pixels at once fader that is slightly faster is added. A program frame added that may be used as a start for new programs (FRAME). **@colour** in TIMER is changed. **@print** is now available in SHRTONES.

3.5 Version 3

True colour fade routine added (found in the jagdemo).

3.6 Version 2

This one has got a file that is necessary to use DSPMOD and a file that is necessary to use TRIG included, unfortunately I forgot them in v1. A new function in WINDOW, **@loadrsrc** loads a resource file and creates a window with the loaded object in it.

3.7 Version 1

The first one. Lots of files missing.

4 Credits

4.1 DSPMOD

DSP MOD replay routine written by bITmASTER of BSW. Additional code by Dusan.

4.2 GORAUD

Fast Gouraud shading polygon routines, July 1993 by Griff.

4.3 CACHE

Cache equates by Griff.

4.4 FADE

Fadeit - true colour versh, coded by ¡Genie!¿ of Network Trash.

5 Address

Most of the other stuff and putting everything together was made by Johan Karlsson. Any contributions are very welcome, my address is:

email: d92jk@efd.lth.se

SnailMail:
Magistratsv. 55 X112
226 44 Lund
SWEDEN

Telephone: (0)46 392017