

Brief overview of all of the facilities in Cfunctions.h

(by Peter Mather / posted @TBS 27 February 2016/V0.1)

The best way to understand this is to obtain the Micromite 5.1 source from MMBasic.com. Load it into MPLabX and then you can use "find in projects" in the "edit" menu to understand properly how things work. Geoff's code is well commented and beautifully structured so easy to interpret.

Basically there are 3 categories of interfaces to the Micromite firmware available

1. Micromite firmware routines that can be called from CFunctions
2. Definitions that allow CFunctions to use Micromite firmware data
3. Definitions that allow the Micromite firmware to call CFunction routines

The latter is the most complex so I'll leave that to last:

Micromite firmware routines that can be called from CFunctions

General calls

int a = CurrentCpuSpeed; // returns the CPU speed in Hz

uSec(a); // waits for "a" microseconds (automatically compensates for CPU speed)

putConsole('x'); // writes a single character (char) to the console

char a= getConsole(); //reads a single character from the console, returns -1 if no character in buffer

ExtCfg(pin, mode, options); //SETPIN command, valid parameters are given at the bottom of Cfunctions.h

ExtSet(pin, value); //checks if a pin is a digital output and sets it to "value" if it is

int a = ExtInp(pin); //Get the value of an I/O pin and returns it. For digital returns 0 if low or 1 if high. For analog returns the reading as a 10 bit number with 0b1111111111 = 3.3V

PinSetBit(pin,IOreg); //sets the pin's bit in the specified I/O register. The list of IO registers is given as the second last section in CFunctions.h. Can be used for many functions e.g.
PinSetBit(pin,LATCLR) == PIN(pin)=0

int a = PinRead(pin); //returns the value of a digital input pin (0 or 1)

int a = GetPortAddr(pin, IOreg); //gets the hardware register address of the defined register for the pin specified. Used to implement fastest possible I/O from a CFunction

int a = GetPinBit(pin); //gets the position in the I/O register of the pin. Use $1 \ll \text{GetPinBit}(\text{pin})$ to get a mask for that pin

MMPrintString(s); //Prints on the console the string pointed to by "s" (e.g. char s[10]). Can only be used with string literals by using the technique identified in the earlier post in this thread

char a[10]; IntToStr(a,number,base); // converts the integer "number" to a string in "a" using base "base"

char a[10]; char padchar = ' '; FloatToStr(a,number,charsbefore,charsafter,padchar); //converts the floating point number "number" to a string in "a" with "charsbefore" characters before the decimal point and "charsafter" after it. Pads leading array elements with the char padchar.

CheckAbort(); //returns to the command prompt if ctrl-C pressed, otherwise processing in the CFunction continues

char *p = GetMemory(nbytes); // gets some Basic memory for use in the Cfunction. The memory is available all the time the program is running. In the example the memory can then be accessed as an array p[0] to p[255]

char *p = GetTempMemory(nbytes); // gets some Basic memory for use in the Cfunction. The memory is returned when the Cfunction exits even though the program may stay running. In the example the memory can then be accessed as an array p[0] to p[255]

FreeMemory(p); //returns the memory pointed to by "p" to Basic

SoftReset(); // Executes a reset of the Micromite

TFT drawing calls

DrawRectangle(x1, y1, x2, y2, colour); //Draws a rectangle between x1,y1 and x2,y2 filled in colour. NB DrawPixel is the same as DrawRectangle where x1,y1=x2,y2 and is defined as a macro. Take care when using this as, for example, DrawPixel(X++,Y++, colour) will have unintended consequences

DrawBitmap(x1, y1, width, height, scale, fc, bc, bitmap); // Draws a bitmap starting at x1,y1 of width and height specified, scaled by scale with bit set to 1 in colour "fc" and bits set to 0 in colour "bc".

DrawLine(x1, y1, x2, y2, width, colour); // Draws a line between x1,y1 and x2,y2 of the "width" specified and in the "colour" specified

int a = HRes; // returns the horizontal resolution of the screen

int a = VRes; // returns the vertical resolution of the screen

Floating point calls

float c = FMul(a,b); //multiplies a and b

float c = FAdd(a,b); //adds a and b

float c = FSub(a,b); //subtracts b from a

float c = FDiv(a,b); //divides a by b

int c = FCmp(a,b); //returns 1 if a > b, returns 0 if **a = b**, returns -1 if b > a

float c = FSin(a); // returns sine of angle "a" specified in radians

float c = FLog(a); // returns the natural logarithm of "a"

float c = FPow(a,b); // a raised to the power b

float c = atanf(a); // returns atan(a)

int c = FloatToInt(a); // returns "a" converted to an integer

float c = IntToFloat(a); //returns "a" converted to a float

float c = LoadFloat(0xnnnnnnnn); //converts the hex representation of a float into c.

Definitions that allow CFunctions to use Micromite firmware data

datatype element c = Option->element; // Used to read an element from the Option structure

Option->element = a; // Used to write an element in the Option structure

The option structure is given in CFunctions.h. It can be read and written from within a Cfunction. Great care should be taken if writing to understand the implications. It is used extensively in loadable drivers to save and read the pin numbers for CS, CD, RESET etc. The Option structure supports the various Basic OPTION commands.

int c = ExtCurrentConfig[pin]; //Used to interrogate the current I/O mode of a pin. Valid entries are given at the bottom of CFunction.h

int c = CFuncRam[n]; // CFuncRam is a 256 byte array (64 integers/floats or 32 long long) area of memory exclusively reserved for CFunctions . It is not changed by RUN unlike Basic memory so can be used for storing information that subroutines in a CFunction may need that has been set up by an initialisation routine run, for example, by MMSTARTUP. If you want to use it as an array of a datatype other than integers use the syntax.

char *p = (void*)CFuncRam; char c = p[n]; // allow access to CFuncRam as a char array

Definitions that allow the Micromite firmware to call CFunction routines

This is the most complex set of CFunction mechanisms but also the most powerful. In order to use these we need to know where in memory the CFunction is located. This can be done by passing into the Cfunction its address obtained by:

address = peek(CFunAddr, cfunctionname)

and then adding an offset from the main CFunction to the function to be called by the Micromite firmware. Alternatively, the boiler plate presented in the posts above does all this for you:

```
void routinetobecalled(void){
}

__attribute__((noinline)) void getFPC(void *a, void *b, volatile unsigned int *c)
{
    *c = (unsigned int) (__builtin_return_address (0) - (b -a)) ;
}

long long main(void){
    volatile unsigned int libAddr ;
    getFPC(NULL,&&getFPCLab,&libAddr) ; // warning can be ignored
    getFPCLab: { }
    MMFirmwarevector=(unsigned int)&routinetobecalled + libAddr;
}
```

Given this the actual function pointers we can set are as follows:

TFT drawing calls

Specifying just two CFunction routines allows us to implement a loadable driver for any type of display.

void MyDrawRectangle(int x1, int y1, int x2, int y2, int c)
DrawRectangleVector=(unsigned int)&MyDrawRectangle + libAddr;

This sets the function that the Micromite firmware will call if trying to draw a rectangle. The calling sequence must be exactly as specified.

```
void MyDrawBitmap(int x1, int y1, int width, int height, int scale, int fc, int bc, unsigned
char *bitmap )
DrawBitMapVector=(unsigned int)&MyDrawBitmap + libAddr;
```

This sets the function that the Micromite firmware will call if trying to draw a bitmap (e.g. TEXT). The calling sequence must be exactly as specified.

There is one additional routine that may be specified in a driver if you wish to use the display as a console device:

```
void MyScrollLCD(int lines)
ScrollLCD=(unsigned int)&MyScrollLCD + libAddr;
```

I haven't attempted to use this so far.

Timer/repeat calls

The next set of vectors allow Cfunction routines to be called repeatedly.

```
void TickInt(void); //called every millisecond in the MM firmware clock routine
CFuncmSec=(unsigned int)&TickInt + libAddr;
```

```
void BasicInt(void); //called after every Basic Statement
CFuncInt=(unsigned int)&BasicInt + libAddr;
```

```
void Timer1Int(void); //called by the timer1 interrupt routine
// The main CFunction is responsible for setting up the timer to interrupt as required. NB using
this interrupt is incompatible with also using IR
CFuncT1=(unsigned int)&Timer1Int + libAddr;
```

```
void Timer5Int(void); //called by the timer5 interrupt routine
// The main CFunction is responsible for setting up the timer to interrupt as required. NB using
this interrupt is only available on the MMPlus
CFuncT5=(unsigned int)&Timer5Int + libAddr;
```

(Source: http://www.thebackshed.com/forum/forum_posts.asp?TID=8389&PN=1)