Introduction

Those of you who wish to learn about ARM microcontrollers may consider a nice little board "The MINI-M4 for STM32". This is a small board implemented as a DIP40 and features a STM32F415RG microcontroller and at $29 US it is very affordable. The only other piece of equipment required is a USB cable with a Type-A connector on one end and a Mini-B connector on the other end which also supplies power to the board. The microcontroller comes with a bootloader already programmed into the chip so that a programmer is not required.

Hardware

The board contains very few auxiliary components. There are 2 programmable LEDs, one yellow and one red. The green LED is connected to the 3.3 volt power rail. A 16 MHz crystal supplies the clock to the microcontroller that can be boosted by the PLL up to 168 MHz. A 32.768 KHz crystal supplies the oscillator for the Real Time Clock (RTC). Due to the lack of space, the battery backup pin for the RTC is simply wired to the 3.3 volt rail. Which means that the RTC looses its time setting when power is removed from the board. There is a Reset pushbutton and 36 I/O pins neatly brought out to the edge of the board. Most importantly, the board includes a USB socket already wired to the appropriate pins of the microcontroller. This interface will be used for program load and for communicating with the microcontroller.

An incredible number of peripherals are fully integrated inside the microcontroller chip. These are accessible through memory mapped registers. It takes a little bit of programming to make use of these peripherals. It's best to start with the simplest one and keep adding software modules as the need arises. If you have the compiler, Mikroelektronika has supplied a Library of useful function calls that can make life easier for beginners. However, I like to use as little as possible of those libraries and program the control registers directly. That way, I can be more intimate with the hardware and learn what each bit in a register does. I also dislike that the code is not published and I don't know exactly what it does. Reading through the Reference Manual and the Datasheet is essential. Don't be dismayed by the fact that the Reference Manual for the microcontroller alone has 1422 pages.

Software

The USB_Driver.c file uses the Mikroelektronika library that implements USB communications and puts a wrapper around it. For convenience to the user, outgoing messages on the USB interface can be queued for sending by placing a pointer to the message into a recirculating buffer. This is done automatically by a call to USB_PutQueue(char *msg). For example, if you have some text you want to send to the PC application, you just call the function directly: USB_PutQueue("Hello world"); If you have some variables that require formatting, you have to create a buffer in RAM using the sprintf function. For example: sprintf(MyBuffer,"Variable value: %d”,MyVariable); followed by USB_PutQueue(MyBuffer); You will get the hang of it once you read some of the code provided. I have included a module that defines a bunch of buffers that can be used for this purpose. Define a variable char *BufNum; and in your code call BufNum = GetBuffer(); This will get you a pointer to the next free buffer and you can use it for all sorts of things. You will see code in many places like this:

```
BufNum = GetBuffer();           // Get a free buffer
sprintf(BufNum,"%c\r\n",readbuff[0]);
USB_PutQueue(BufNum);
```

The receiving side decodes simple commands and performs actions on those commands. A few commands have been implemented as follows:

a – responds with “Command a received” as a test to make sure it’s working OK
b – sets the yellow LED
c – clears the yellow LED
d – sets the red LED
e – clears the red LED

Then we add a file for the Real Time Clock. USB_Driver.c now includes 4 more commands:

f – sets the RTC to the date and time which follows the command. For example – f,14,04,01,02,18,44,36 sets the date to Year=14, Day of week= 4 (Thursday), Month=01 (January), Day=02, Hour=18, minute=44, and Second=36
g – displays the current RTC date and time. You should see the seconds increasing as you repeatedly send this command.
h – writes some data to RTC backup registers. If there was a battery attached to the microcontroller Vbat pin, data in these registers would be preserved even when main power was removed.
i – displays the Initialization and Status Register.

Finally we add an example that uses DAC1 to generate a Sine wave at 60 Hz. To do that I will use a Sine lookup table and assign 1000 data points to one cycle of the Sine wave. In order to provide a time base for DAC1 I have to introduce some timer resources. These will be in Timers.c file. Timer6 will drive DAC1 loading. Upon timeout of timer6, it will generate an interrupt where the next DAC1 value will be obtained from the Sine table. Before it is loaded into the DAC, it will be multiplied by an amplitude value (0 to 255) and divided by 255. This has an attenuating effect on the value in the Sine table. When the amplitude value is 255 (at maximum) then the attenuating value is 1 (255/255) and the DAC is loaded with the value from the Sine table. When the amplitude is 127 (mid point) the attenuating value will be 127/255 = 0.498 or approximately ½. When the amplitude is 0 then the output should be a flat line. The output of DAC1 is available on I/O pin PA4 for scoping.

The Sine table only holds ¼ of the data points since the 4 quadrants of a Sine wave can reuse the data. The address where the data is fetched from the Sine table starts at 0 and is incremented after each interrupt. The address is checked after each access and when the code detects that the address is at 250 (251st element of the table), the direction is switched so that each subsequent interrupt decrements the address. The address then decrements until the code detects address 0. At that point the address is incremented again and so on.

A simple calculation will show that 60 Hz and 1000 data points results in 60,000 interrupts per second with a multiply and divide for each one. This a pretty high rate and I wouldn't use it like this in an application but I wanted to see if it would keep up. I have included a spreadsheet that can calculate a smaller Sine table.

There are 3 commands in USB_Driver.c that deal with this example. They are:

j – initialize DAC1 and Timer6
k – raise the amplitude of the Sine wave
l – lower the amplitude of the Sine wave

What's Next

This example only scratches the surface of what is possible with this little board Additional modules could be written to support the following peripherals:

General Purpose Input Output (GPIO)
Direct Memory Access (DMA)
Analog to Digital Converter (ADC)
Many more Timers (TIM1 to TIM14)
Independent Watchdog Timer (IWDG)
Random Number Generator (RNG)
Controller Area Network (CAN)
Inter-integrated Circuit (I2C)
Universal Synchronous Asynchronous Receiver Transmitter (USART)
Serial Peripheral Interface (SPI)
Etc.

Keep in mind that these modules are easily ported to other boards with an ARM processor.