Occam’s Microcontroller

Paul Darlington - m0xpd - 8 Uplands Rd, Flixton, Manchester

It might be supposed that the internal complexity of microcontrollers, digital synthesizers and similar devices places them somewhat at odds with our guiding principal of “*economy of means allied with richness of result*” [1]. It is, however, the purpose of this article to argue that recent developments in microcontroller technology and the ready availability of other digital subsystems have made these resources an efficient foundation for the development of minimalist radios. Further, these developments have been such that potential benefits are accessible to all experimenters, rather than just those who have special interest or expertise in digital methods.

Microcontrollers are “small computers on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals”. They are intended for *embedding* into larger systems, where they serve as controllers and communication channels. They have been used in peripherals of the rigs featured in these pages for several years, serving in roles such as keyers, frequency displays, power & SWR displays, etc. However, this article will explore by example the idea of embedding the microcontroller right into the heart of a QRP rig, rather than being content to leave it on the periphery.

To date, two important obstacles have limited penetration of embedded computing into the working vocabulary of home-brew QRP enthusiasts; complexity (real or perceived) and inflexibility (especially those designs presented in “closed-source” format). We have seen laudable attempts to demystify microcontrollers, including in these pages [2], but those initiatives have not yet overcome the barriers to widespread adoption.

Winds of Change

Fortunately, a quiet revolution has been taking place in the world of microcontrollers, embedded systems and “physical computing”. This revolution was not targeted at technically sophisticated beneficiaries – rather it was intended for artists, hobbyists and schoolchildren. Radio amateurs, with their explicit technical skills proven by licensing, are well equipped to thrive in this post-revolutionary “new world”.

The quiet revolution has not been achieved through change in the microcontroller devices themselves – they still express the same internal architecture and still offer the same features as before, albeit at continually falling cost/performance ratio. Rather, the devices have been presented in different contexts, designed to make it easy for users to exploit their power through accessible, intuitive programming languages, easy to use (and usually free) development environments and powerful, flexible, inexpensive and standardized hardware platforms. All of these make it easy to get a microcontroller to do a useful job of work – like form the backbone of a QRP rig.

The opening steps in this revolution have been played out in full sight of G-QPR members, who saw the original PIC microcontroller [2] “re-cast” as the “PIC-AXE”, which was featured at a recent G-QRP mini-convention [3] and is available through club sales. The

PIC-AXE *is* a PIC - but a PIC that has been tamed by the provision of new, simpler means to interact with it and exploit its features. There is a clearer (*and much more successful*) embodiment of this revolution in the world of the AVR brand of microcontrollers, in which the ATMega device has been "re-cast" as the "Arduino".

Arduino is more than a subset of the AVR microcontroller, underpinned by software resources. It also has physical expression as a family of boards, making it truly an "electronics prototyping platform". These boards bring the resources of the microcontroller into easy reach, making it laughably easy to connect to systems such as our radios – as we shall see. A board-level device called the "Pinguino" is an analogous system for the PIC microcontroller.

Most importantly of all, the revolution in microcontroller accessibility is open–source, both in software AND hardware. This makes it not just possible but actively encouraged to share and build on the efforts and experience of a vast community of users.

The argument of this article is agnostic to the particular brand and family of microcontroller used. However, we shall present all subsequent examples in the context of the Arduino platform.

Manna from Heaven

In addition to developments in the way microcontrollers are supported, described above, an interesting crumb has fallen from the table of high-tech electronics. Opportunistic QRP enthusiasts have been quick to pick up this crumb – which takes the shape of a "Direct Digital Synthesis" Signal Generator Module, today widely available for less than the price of the (AD9850) chip it contains. With this module it is possible to generate stable, controllable sinusoids "from dc to light" (or, at least, to beyond the top of the HF band). All that is required is some means to control the module and communicate with it – a perfect application for a microcontroller.

Commenting on his QRO power and SWR meter, [4], Ray, g4fon, observed "*that meters have become expensive, whereas an LCD display might ... actually be less expensive and more robust*", fully justifying the use of digital building blocks. Similarly, we should notice that a microcontroller and a DDS module is cheaper than a decent quality variable capacitor and reduction drive, such that a digital VFO is now not just a feasible technology but also a literal "economy of means". However, controlling either a SWR meter or a VFO will leave a microcontroller twiddling its thumbs for most of the time. It could undertake both tasks - and more...

If we assemble a partial list of the features within a practical HF CW transceiver that could be managed by a microcontroller, the possibilities are impressive:

Keying Iambic Keyer Automated “CQ” calls Beacon Operation	**Display** Frequency SWR/Power Field Strength
Frequency Input from dial / keyboard / memory	**Tx/Rx Switching** Tx Keying Rx Muting QSK / Semi break-in delay
Frequency Control DDS Control Tx/Rx Offset and “RIT”	

Notice that some items of this list are possible only because of the use of a digital device (*management of a DDS chip or module for the local oscillator, digital frequency input and display, etc*). Other items of the list might previously have been interpreted as functionality normally residing outside the rig (*such as the keyer*). Use of the microcontroller allows us to extend the scope and ambition of our homebrewed rigs and make them more complete as self-contained communication systems - like commercial rigs. However, there are items on our list that are “logical” (i.e. essentially DIGITAL) tasks, which would have been required in the rig whatever technology was used. This is particularly seen in the management of transmit and receive switching. QRP rigs have always included such switching functions and their implementation in a microcontroller will be found to add considerable flexibility.

Obviously, we could extend the list to add any number of “higher” functions (such as automatic adjustment of an ATU) – but we shall deliberately stay well clear of the slippery slope that leads towards software defined radios and similar complexity. Our interpretation of “economy of means” requires that we “keep it simple”.

Example: A QRP CW Transceiver

We now present an example of a practical and successful CW rig, built on the principles outlined above. The rig has made CW QSOs on the 80, 40 and 30 metre bands and has operated as a beacon in QRSS and various FSK modes. It is not the purpose of this article to be proscriptive, so the details of the rig are not important. Instead, we shall focus only on those points of connection and interaction between the microcontroller and the rig. The building blocks of the rig are typical of ordinary QRP practice and so these “points of connection” may provide inspiration for readers to modify their own favourite circuits for micro-control.

The overall architecture of the example rig is seen in the figure over the centre pages, which includes details of the interface between the rig’s sub-systems and the microcontroller, discussed below.

Keyer

It is useful to start with the iambic keyer – for two practical reasons. Firstly, a keyer is a very useful “first project” for those new to microcontrollers, expanding on the traditional “press a button and light an LED” application, common to most microcontroller training courses. Secondly, the keyer actually will form the “backbone” of the software for our example rig.

The keyer’s hardware is trivial. One side of each switch of the key and/or paddle is connected to an input pin of the microcontroller, with the other side of the switches commoned to ground. Most microcontrollers feature internal pull-up resistors, which can be enabled on certain inputs. These pull-ups hold the input “high” until the key is pressed, which action will short the input down to the alternative “low” logic state. The key’s state is sensed by sampling the voltage on the input pin repeatedly in the main operating loop of the software and watching for a change between input voltages on two successive observations. A change from “high” to “low” voltage indicates a key press and the opposite transition indicates a key release. The time taken between observations of the input (time during which the microcontroller is attending to other tasks) creates a switch de-bouncing delay.

It has been found possible to build not only the keyer but also the entire transceiver without using the “interrupt” facilities of a micro-controller, thus keeping it simple. The resulting rig does not have any noticeable latency between operating the key or paddle and hearing the sidetone (which itself is generated by a native function of the Arduino language).

The same code which implements an iambic keyer can easily be expanded to read a message saved as text, one character at a time, look up the Morse equivalent of each character and send it at a pre-determined speed, making possible automated CQ calls and beacon operation.

Local Oscillator

The combination of a DDS chip (or module) and a microcontroller make possible a flexible RF generator with range, stability and accuracy that matches or surpasses conventional analog oscillators traditionally used in our rigs. We shall examine the hard- and soft-ware required to make a flexible VFO in a moment but, for now, we shall consider what is required just to set the local oscillator running at one frequency. The DDS chip typically will expect to receive configuration information from a controlling device in ‘serial’ format. This is to be preferred over the ‘parallel’ alternative as the number of available pins on any microcontroller is limited.

In the case of the Silicon Labs’ Si570 device (used in the excellent USB synthesizer [5] and pa0klt VFO [6]) translation between the desired frequency of operation and the configuration code required to produce this is not simple. However, the AD9850 device accepts a numerical input proportional to the desired frequency, making programming very easy...

Here’s the Arduino code (developed from [7]) required to calculate the configuration data and send it to the AD9850...

```
// calculate and send frequency code to DDS Module...
void sendFrequency(double frequency) {              // ‘frequency’ is the desired f
  int32_t freq = frequency * 4294967296/125000000; // module has 125MHz xtal
  for (int b=0; b<4; b++, freq>>=8) {               // ‘freq’ has 4 bytes...
    shiftOut(DATA, W_CLK, LSBFIRST, freq & 0xFF);   // sent one at a time
  }
  shiftOut(DATA, W_CLK, LSBFIRST, 0x00);            // final “0” control byte
  pulseHigh(FQ_UD);                                 // and a pulse on FQ_UD
}
```

The DDS module hosting the AD9850 offers both a parallel and a serial interface – of course, we choose to use the serial interface, which corresponds to the pins named “DATA” and “W_CLK” in the code segment above. Other connections (for power and two control lines, FQ_UD and RESET) are shown in the main figure. All connections to the module are through an easy-to-use 0.1 inch header – which strongly differentiates this module from the rather less user-friendly physical interface to surface-mount DDS chips.

Keying the Tx

With a local oscillator generating a stable source of RF and means to generate the timed sequences of Morse code, all that is required to transmit CW is means for the microcontroller to key the transmitter. A single transistor, in open collector configuration, replacing the physical “key” in a conventional QRP transmitter design, easily achieves this. The Tx detail in the main Figure shows an example of this approach applied to an early stage of the “Ugly Weekender” [8] transmitter.

The building blocks described to this point - keyer, oscillator and means of keying the power stage - add up to a simple CW transmitter or a QRSS beacon. This transmitter or QRSS beacon is more flexible than the traditional rock-bound alternative, as it can QSY to any frequency with a simple re-flashing of the microcontroller’ program. However, the full potential of the microcontroller-based rig only begins to be unleashed once we exploit some of the dynamics of the DDS module and add a receiver…

Varying the Oscillator

The DDS module reacts very quickly to a command to change frequency, with negligible disturbance of the output waveform at the instant of transition (indeed, the transition is phase-continuous). This, coupled with the speed of the microcontroller, makes the oscillator frequency-agile, facilitating FSK modes (such as WSPR [8]), the transmit/receive offset inherent in CW operation and frequency changes to tune across bands and switch between them.

The simplest means to control the frequency of a DDS local oscillator is to read the voltage on a potentiometer wiper, configured as a potential divider between Vcc and ground. Most microcontrollers feature integral analog-to-digital converters. These are capable of directly reading such a voltage to (typically) 10-bit resolution (*i.e. they are able to resolve to ~ 0.1% of full-scale-value*). Thus, with the addition of a single, simple potentiometer, we can make the previously rock-bound transmitter described above able to tune over useful segments of a single band. For example, a 5kHz section of the 40m band can be covered with tuning that sounds and feels continuous in "pitch". This is sufficient to cover both the QRP and FISTS frequencies of interest to many readers.

Such a simple means of adjusting the frequency of the local oscillator will be entirely familiar to QRP enthusiasts, who are used to VFOs with single-knob tuning from a variable capacitor (or potentiometer biasing a varactor diode). The knob provides not only the means to adjust frequency but also a visual indication of the frequency setting. If we wish to fully exploit the flexibility of the DDS module, we must be able to monitor the frequency setting with greater resolution. This can be achieved by a link from the rig to a PC, as the Arduino offers a pair of pins dedicated to such a link and the software tools to send the current frequency to a PC with a single line of code…

However, in requiring the presence of a PC, such an approach is neither minimalist nor always convenient. Fortunately, the microcontroller can easily drive a display…

Display

A numeric or alphanumeric display can provide useful information about the state of a rig, such as the power / SWR monitoring function already mentioned. However, we shall consider the display of frequency as the only necessary function and use it as sufficient example of the integration of a display into a micro-controlled rig. As Ray, g4fon, observed [4], LCD displays are robust and inexpensive and entirely suitable for monitoring the frequency of our DDS module's emissions. There is, however, one obstacle to their use.

Many alphanumeric displays are built using an interface associated with Hitachi, which has an 8-bit data bus and some further control lines to latch data into and out of the device. As already has been mentioned, microcontrollers have only a limited number of input/output pins and this Hitachi interface is expensive in these important resources. Fortunately, there are a number of means by which we can overcome this potential obstacle.

First, it is always possible to upgrade the choice of microcontroller or microcontroller system to increase the number of available I/O pins (e.g. changing from Arduino UNO to Arduino MEGA) – but this approach is the antithesis of minimalism! Second, it is possible to add simple shift registers to make serial-to-parallel converters, by which a small number of I/O pins (on the microcontroller side) may be connected to a large number of peripheral pins – but this approach only really starts to deliver benefits for a large numbers of pins. Third, we can use the Hitachi interface in an alternative "4-bit" mode, in which commands and data are transferred a "nibble" (i.e. half a byte) at a time, saving 4 pins. Fourth, we can use (or, better still, make) a serial-to-parallel converter module, which accepts a two-wire input, such as I2C, from two pins of the microcontroller and connects to the Hitachi interface. Such modules are now widely available for a few pounds.

A fifth approach was used in the development of the rig used as example in this article, which exploits a 12-character numeric LCD display with serial input, found in the junk box! The data line is actually shared with the data line to the DDS module, further reducing the load on microcontroller I/O pins.

Rotary Encoder for QSYs

The finite range of adjustment available from a potentiometer is radically expanded by use of a rotary encoder, which generates digital outputs on rotation of a familiar "knob". These digital outputs can be interpreted by the microcontroller to produce a very flexible means of frequency input, allowing adjustment over a wide range at controllable sweep rates or resolutions. Rotary encoders have two internal switches, which switch in quadrature as the shaft is rotated. The resulting two-bit sequence encodes direction and (if required) speed of rotation. The sequence can be read by very simple logic, as demonstrated by this Arduino code segment…

```
// (as part of the main loop structure…)
RotEncA = digitalRead(RotEncAPin);          // read the two rotary encoder pins…
RotEncB = digitalRead(RotEncBPin);

                                            // if there's a state change on input A…
if ((RotEncA == HIGH)&&(OldRotEncA == LOW)){
                                            // AND input B is LOW…
if (RotEncB == LOW) {
                                            // then knob was moved clockwise…
  Frequency = Frequency + df;}
else {
                                            // otherwise, moved anti-clockwise…
  Frequency = Frequency - df;}
}
OldRotEncA=RotEncA; // save the input A value for next time round the loop
```

We can change the frequency increment *("df" in the code above*) to give an appropriate rate of frequency control; df=10Hz is small enough for "continuous pitch" tuning of CW, whilst df=1kHz gives a reasonably quick sweep across the entire band. Many rotary encoders feature a third internal switch, operated by a push on the knob. This can be used to toggle through a set of different values for the frequency increment, df. The example rig uses 10, 100 and 1000 Hz.

With frequency inputs passed from a rotary encoder to the DDS and monitored by an LCD display, the oscillator becomes a truly flexible VFO, which can tune the transmitter described above over multiple bands.

Transmit Muting and Break-In

The same logic signal used to key a CW transmitter can be used to simultaneously mute a receiver, making an entire transceiver. The interface between a microcontroller I/O line and a practical Rx mute is shown, in the example rig, in the context of the FET passgate used in the receiver of Roy Lewallen, w7el's "Optimised Transceiver" [9]. Although it is possible to share the Tx keying and Rx muting on a single I/O line, having these functions separated allows greater flexibility in managing break-in delays.

In the example rig supporting this article, keying the transmitter initiates a counter, which implements a fixed delay. The receiver is muted for the whole of this delay period. The same delay manages the frequency offset required between Tx and Rx. If the transmitter is keyed again during the delay interval, the count is re-started. The start value of the count determines the mute interval – a very small count value gives full break-in, whilst a larger value holds the receiver muted over individual Morse characters or words.

The potentiometer used for simple “analog” frequency input can be exploited as a “Receive Incremental Tuning” input to vary the transmit offset, where main frequency input is via a rotary encoder.

Code Samples, Schematics and a Note of Caution
Although the example rig supporting this article was intended to illustrate methods and inspire others to experiment with their own microcontroller-based QRP systems, there may be further details of its construction and code of interest to readers. A full schematic and a series of Arduino programs (“sketches”) are available for download at:
http://www.gqrp.com/sprat.htm

To date, only one downside of the example rig has emerged; the DDS module draws 200mA, which clearly limits its usefulness for /p operation.

Whilst many QRP enthusiasts enjoy using anachronistic or even obsolete technologies in their h/b rigs, the microcontrollers and other digital resources used in commercial transceivers are now available and accessible for widespread exploitation. This digital technology can offer “economy of means allied with richness of result”.

Notes and References

[1] D L Sayers paraphrases Occam’s Razor in these words in her novel
“The Five Red Herrings”, 1931
[2] “Starting with PIC”, P Debono, 9h1fq, SPRAT 138, p17
[3] “A Maine Yankee in Rishworth Court”, R Harper, w1rex, SPRAT 141, p34
[4] “QRP Wattmeter”, R Goff, g4fon, downloaded from:
http://www.g4fon.net/wattmeter.htm
[5] QRP2000 USB Controlled Synthesizer Kit:
http://sdr-kits.net/QRP2000_Description.html
[6] pa0klt “Low Noise VFO Synthesized Kit”:
http://sdr-kits.net/PA0KLT_Description.html
[7] Testing an eBay AD9850 DDS module with Arduino Uno, R Rollinson, nr8o:
http://nr8o.dhlpilotcentral.com/?p=83
[8] "Ugly Weekender", R Hayward, ka7exm & W Hayward, w7zoi, QST, August 1981 – see also Todd Gale, ve7bpo’s notes at: http://www.qrp.pops.net/transmit.asp
[9] “An Optimised QRP Transceiver for 7 MHz”, R Lewallen, w7el. QST, August 1980 – now available at:
http://www.arrl.org/files/file/Technology/tis/info/pdf/93hb3037.pdf