**Beacon**

**Purpose**

In order for Amateur radio users on the ground to be aware when BLUEsat is flying overhead, the satellite needs to transmit a periodic beacon that can pick up. The current format for the BLUEsat beacon is a Morse code message. This message may take the form of BLUEsat's callsign followed by a short message such as "BLUEsat" for easy identification. The Morse code will be outputted as raw audio through the Switching Circuit and passed to the Satellite's Transmitter periodically.

**Implementation**

The Beacon is split conceptually into the following subcomponents: the **Memory**, the **Counter**, the **Tone Generator** and the **Clock Generator**. These are explained below.

The centerpiece of the Beacon is the **Memory** - a one-time-programmable EPROM chip. The chips we use are in the AT27 family of ATMEL parallel EPROMS, specifically the AT27LV256. It runs on any supply voltage from 3.0V to 5V, takes in a 14-bit address bus and holds 256KBit or 32768 B of memory.   
The memory chip's pinout is as follows:  
1. VPP - the programming voltage. This is tied to Vcc to put the memory in read mode.  
2. A0-A14 - the address bus. As mentioned earlier, A0 - A7 are driven by an 8 bit counter, the other address pins are tied to GND.  
3. O0-O7 - the data bus. O0 is tied to the tone generator's RESET pin and O1 is tied to the Counter's RESET pin. The reason for this explained below.  
4. C\E\, O\E\ are both tied to GND to enable the chip and the output pins.  
5. VCC and GND are tied to a power bus and GND respectively.

The second component is the **Counter**. This is simply a 4520 dual 4-bit binary counter configured into an 8-bit ripple counter configuration(see drawing COMS-!!@#). It's outputs drive the Memory's A0-A7 pins, causing the memory to output bytes from addresses 0x00 to 0xFF.

The **Clock Generator** provides the Counter with a 10Hz square wave signal. This causes the Counter to step from address to address every 100 ms. The choice of clock frequency is predicated on making the Morse code message playback slow enough to be easy to decipher by a prospective radio user.The Clock Generator is a simple design - a 555 timer configured into an astable configuration. (For a detailed schematic, see drawing COMS-!@#%)

The final component is the **Tone Generator**. This provides an audio frequency (specifically, 1kHz) squarewave that can be sent to one of the onboard transmitters.  
The Tone Generator uses the same design as the clock generator, modified to produce a higher frequency squarewave. Also, whereas the Clock Generator's RESET pin is permanently tied high, the same pin in the Tone Generator is tied to the O0 output of the memory. Thus, when the memory output O0 is high, the Tone Generator will produce a 1kHz tone. When O0 is low, the Tone Generator is "silent".  
The output 01 is connected to the RESET inputs of the Counter. It's purpose is to reset the address to 0x00 after each time the transmission ends. This is elaborated on in another part of this document.

**Memory Programming**

In order for the transmitted Morse code message to be "legible" to a radio operator trained in deciphering Morse, it must follow a certain standard. That is, each symbol - the dot ("dit") dash ("dah") intersymbol pause (the gap between "dits" and "dahs") interletter pause and interword pauses must go on for prescribed lengths of time. These intervals are described in the International Morse Code Standard. This Standard defines a "time unit" - this is the length of a single "dit". The other symbols' intervals are actually multiples of the "dit" interval: Thus, if a "dit" is 100ms of tone, a "dah" will be 300ms of tone, an intersymbol pause will be 100ms of silence, an interletter and interword pause will be 300ms and 700ms respectively.  
There is, therefore, a straightforward way to map a Morse Code message to a binary stream, using "1" as 100ms of tone and "0" as 100ms of silence.  
Using this method to encode "SOS S" will give us "1 0 1 0 1 000 111 0 111 0 111 000 1 0 1 0 1 0000000 1 0".   
Although the EPROM we use has 8-bit outputs, we only need one pin to output a sequential binary stream like the one above. We used "1" and "0" ASCII symbols to encode 1 and 0 bits respectively and 0xFF to encode the end of our message. This way, the O0 pin on the memory will output a 1 on an ASCII "1" and a 0 on an ASCII "0" and O1 will output a 1 when the message ends, causing the Counter to reset it's address to 0.  
Therefore, if we wished our Beacon to output "SOS" -> "1010 1000 1110 1110 1110 0000"repeatedly, we'd program the memory this way:

0x00 | 0x31 0x30 0x31 0x30  
0x04 | 0x31 0x30 0x30 0x30  
0x08 | 0x31 0x31 0x31 0x30  
0x0b | 0x31 0x31 0x31 0x30  
0x10 | 0x31 0x31 0x31 0x30  
0x14 | 0x30 0x30 0x30 0x30  
0x18 | 0xFF 0xFF 0xFF 0xFF

**Development History**

Naturally, the first step was to construct a breadboard-based prototype. Although this prototype was functional in principle, it was also extremely fragile, prone to mechanical failures and bulky. Therefore, we chose to create a PCB design of the device. This design did work just as well as the breadboard prototype, it still had a few flaws: it was relatively large(caused by the DIP footprint of the Memory), it worked only at 5V or more (caused by the 555 timers used) and it had no power supply noise decoupling. Also, the message playback wasn't entirely accurate. This inaccuracy was caused because of the double role of the O3 pin on the 4520 timer as both the Memory address input and the clock signal to the second half of the 4520. Thus, when the reset signal would trigger, the O3 pin would go from high to low and clock the second half of the Counter , causing the address to jump to 0x10 rather than 0x00. This was rectified by stalling the reset in memory until an address not using O3 is reached.  
The current version

* reduces the size using SMD rather than through-hole components
* enables 3.3V operation by using CMOS equivalents of 555 timers
* introduces power supply decoupling capacitors on each IC