Hellschreiber FELD_HELL

FELD_HELL transmits glyphs (ASCII characters or could do Chinese, Korean, etc, if they can be represented in the pixel matrix). The glyphs are pixels in 7 columns of 14 half-pixel rows.

The Glyph Table is an array of a `Glyph` structure like this

```
typedef struct glyph {
  char ch ;               // the character represented, 'A'...'Z'
  word col[7] ;           // the 7 columns of 14 bit pixels in 16 bit words
} Glyph ;
```

The structure gives access as `glyphtab[index].ch` and `glyphtab[index].col[0-6]`.

A character uses 7 columns, each hexadecimal number contains the pixels for 14 pixels of each column, read from the `word, b0` to `b13`

```
Glyph glyphtab[] = {
  {' ', {0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000}},
  {'A', {0x07fc, 0x0e60, 0x0c60, 0x0e60, 0x07fc, 0x0000, 0x0000}},
  {'B', {0x0c0c, 0x0ffc, 0x0ccc, 0x0ccc, 0x0738, 0x0000, 0x0000}},
  {'C', {0x0ffc, 0x0c0c, 0x0c0c, 0x0c0c, 0x0c0c, 0x0000, 0x0000}},
```

for example the columns of the letter '`B`' for the 14 binary bits `b0 - b13` are vertically as half-pixels ('`.`' = pixel off, `x` = pixel on)

```
. . . . . . .
. . . . . . .
x x x x . . .
x x x x x . .
. x . . x . .
. x . . x . .
. x x x . . .
. x x x . . .
. x . . x . .
. x . . x . .
x x x x x . .
x x x x . . .
. . . . . . .
. . . . . . .
```

So there are two half-pixel rows blank at the top and bottom, and two complete columns blank at the right, for inter-glyph spacing.

Timing

FLED-HELL has exact timing, pixels for each glyph are read as columns left to right, and in each column bottom to top, and each complete glyph is sent in 400ms, which is 57.14ms per column, or for 14 rows, 4.0816ms per pixel. This means the transmit rate is 1/(2 x 4.0816) = 122.5 baud, a throughput of 2.5ch/sec or 25 wpm. The code below was proposed by NT7S.

The Arduino Timer 1 is used to generate this timing, by programming it to generate an interrupt at 245Hz (every 4.0816ms) for sending each half-pixel.

For Arduino Timer 1 the code is

```
noInterrupts();           // Turn off interrupts.

TCCR1A = 0;               // Set entire TCCR1A register to 0;
                          // disconnects interrupt output pins,
                          // sets normal waveform mode.
                          // We're just using Timer1 as a counter.

TCNT1  = 0;               // Initialize counter value to 0.

TCCR1B = (1 << CS10) | (1 << WGM12); // Set CS10 bit and specify
                          // no pre-scaling of the system clock.

TIMSK1 = (1 << OCIE1A);   // Enable timer compare interrupt.

OCR1A = 0xFF1A;           // Set up interrupt trigger count;
                          // 16MHz clock / 0xFF1A counts == 245.00045938Hz

interrupts();             // Re-enable interrupts.
```

The interrupt service routine simply sets a flag that indicates it time to proceed with the next pixel to send

```
ISR(TIMER1_COMPA_vect) {
  proceed = true;
}
```

The main transmit routine, to send each character `(char c)` of a message, first scans the glyph table `'glyphtab[ndx].ch'` to find the table index `(ndx)` of that character. If it is found the bit mapped glyph is scanned column-by-column (0 - 6) and row-by-row (0 - 13) and either a ‘1’ or a ‘0 is transmitted (RF output ‘on’ or ‘off’).

```
// find index of character in glyphtab then send the pixels
void sendChar(char c) {
```

```
  int ndx, x, y;

  for (ndx = 0; ndx < NGLYPHS; ndx++) {                // scan for index

    if (glyphtab[ndx].ch == c) {                       // if found, send
      for (x = 0; x < 7; x++) {                        // scan 7 cols 0 - 6
        for (y = 0; y < 14; y++) {                     // scan 14 rows 0 - 13
          if (glyphtab[ndx].col[x] & (1 << y)) {       // look for '1's
            dds.output_enable(SI5351_CLK0, 1);         // 1 = pixel on
          }
          else {
            dds.output_enable(SI5351_CLK0, 0);         // 0 = pixel off
          }

          // wait for timer interrupt 4.0816ms (245Hz)
          while (!proceed); // timer interrupt set proceed = true

          // reset flag
          noInterrupts();   // Turn off interrupts; just good practice...
          proceed = false;  // reset the flag for the next character...
          interrupts();     // and re-enable interrupts.
        }
      }
    }

  }
}
```

The output on `CLK0` remains enabled (`dds.output_enable(SI5351_CLK0, 1)) or` disabled `(dds.output_enable(SI5351_CLK0, 0))` until the '`proceed`' flag is set '`true`' by the interrupt service routine, thus meeting the strict pixel timing requirements of FELD_HELL. The proceed flag is reset to `false` for the next pixel.