Technical overview of *I, Robot* video PCB

(For the time being) you can reach me at lordfrito@comcast.net or john.manfreda@idselectronics.com)

Overview of video hardware

The purpose of the video hardware circuit is to take program inputs and convert them to a standard RGB signal. The video hardware contains three main circuits:

- the video object processor
- the alphanumerics processor
- RGB signal generator

The video object processor is basically an rasterizer capable of rendering *DOT*s, *VECTOR*s, and *POLYGON*s. The processor retrieves information from a video object list, rasterizes the objects, and dumps the bitmapped results to one of two banked video screen buffers. The screen buffers represent a bitmap of 256x256 pixels.

The alphanumerics processor takes information from a bank of alphanumerics RAM, and determines which pixels on the bitmapped display need to be activated to display the character.. Each alphanumeric character has a dimension of 8x8 pixels, and the alphanumerics RAM buffer represents a 32x32 character display.

The RGB signal generator combines data from the screen buffer and alphanumerics processor, and converts the result to an RGB signal capable of driving a monitor.

Video object processor

The object list

The video processor takes as it's input a 'recipe' of sorts for creating the screen. This recipe is the object list, which is basically a listing of all objects that need to be rasterized by the video processor. Each object in the list will be rasterized and dumped to the screen buffer. The processor is capable of rasterizing the following object types

- *DOTs* (single pixels)
- *VECTORs* (lines)
- *POLYGONs* (solid filled objects)

The object list begins at address $0000 in COM RAM, and is aligned on WORD boundaries. The value $FFFF, which is reserved, denotes the end of the object list. Each WORD in the object list represents a single object to be rasterized. These values are stored in the following format:

word 0
tt???aaaaaaaaaa

t Type of object to be rasterized
00 =
01 = solid polygon
10 = dot (pixel)
11 = vector (line)

a Offset address of object data list

The video processor rasterizes the object list in the following manner.

```
void RasterizeScreen( WORD * COMRAM )
{
      int address = 0x00;
      WORD object;

      while(COMRAM[address] != 0xFFFF)
      {
            object = COMRAM[address++];
            switch(object >> 14)
            {
            case 1:     RasterizePOLYGON( COMRAM, object & 0x8FF ); break;
            case 2:     RasterizeDOT( COMRAM, object & 0x8FF ); break;
            case 3:     RasterizeVECTOR( COMRAM, object & 0x8FF ); break;
            }
      }
}
```

DOT objects

DOT objects represent a list of pixels to be rasterized into the screen buffers. A single *DOT* object may contain information for displaying multiple *DOTs*, each with a different color.

Information on a *DOT* object is stored in COMRAM at the offset address specified by the object list, and is aligned on WORD boundaries. The value $FFFF, which is reserved, denotes the end of the *DOT* object listing. Every 2 WORDs in the object list represent a single pixel to be rasterized. These values are stored in the following format:

word 0	word 1
xxxxxxxxxxxxxxxx	**yyyyyyyyy?cccccc**

x X coordinate of pixel to be displayed
y Y coordinate of pixel to be displayed
c 6-bit pixel color index value

The video processor rasterizes a *DOT* object in the following manner.

```
void RasterizeDOT( WORD * COMRAM, int address )
{
      WORD word;
      int x, y, color;

      while(COMRAM[address] != 0xFFFF)
      {
            x = COMRAM[address++] >> 7;
            y = COMRAM[address] >> 7;
            color = COMRAM[address++] & 0x003F;

            SetColor( ColorTable[color] );
            DrawPixel( x, y );
      }
}
```

VECTOR objects

VECTOR objects represent a list of lines needing to be rasterized into the screen buffers. A single *VECTOR* object may contain information for displaying multiple lines, each with a different color.

VECTOR object information is stored in COMRAM at the offset address specified by the object list, and is aligned on WORD boundaries. The value $FFFF, which is reserved, denotes the end of the *VECTOR* object listing. Every 4 WORDs in the object list represent a single vector to be rasterized. These values are stored in the following format:

word 0	word 1	word 2	word 3
uuuuuuuuu???????	**vvvvvvvvv?cccccc**	**ssssssssssssssss**	**xxxxxxxxxxxxxxxx**

u 9-bit value representing last scanline that vector appears on
v 9-bit value representing first scanline that vector appears on
s 16-bit value representing number of pixels in each scanline that need to be activated to display line
x 16-bit value representing initial x value

The video processor rasterizes a *VECTOR* object in the following manner.

```
void RasterizeVECTOR( WORD * COMRAM, int address )
{
      WORD  word, s, x;
      int   y, yend, color;

      while (COMRAM[address] != 0xFFFF)
      {
            yend = COMRAM[address++] >> 7;
            y = COMRAM[address] >> 7;
            color = COMRAM[address++] & 0x003F;
            s = COMRAM[address++];
            x = COMRAM[address++];

            SetColor( ColorTable[color] );

            while (y <= yend)
            {
                  DrawLine( x>>7, y, (x+s)>>7, y );
                  x += s;
                  y++;
            }
      }
}
```

POLYGON objects

POLYGON objects represent the information needed to rasterize a solid, filled polygon. Unlike the *DOT* and *VECTOR* objects, a single *POLYGON* object can never render more than one polygon; however, the object format does allow for polygons with a variable degree of complexity. Because of this, the polygon object format is not of a fixed length.

The video processor renders each polygon a scanline at a time, keeping track of starting and stopping scanlines. For any single scanline, two variables are needed to fill the appropriate pixels:

x1 first horizontal pixel on scanline to fill
x2 last horizontal pixel on scanline to fill

To update these two positions for the next scanline, an addition factor must be specified for each of the two endpoints. This addition factor is analogous to slope. The two factors needed are denoted as:

s1 slope / addition factor for **x1**
s2 slope / addition factor for **x2**

Because polygons may have several vertices, these slopes may change as the polygon is being rasterized. The video processor also keeps track of two scanline variables, which alert the processor when a specific slope needs to be updated:

y1 scanline to change value of **s1**
y2 scanline to change value of **s2**

Because the variables **s** and **y** are related to each other, whenever a new **s** value is read the corresponding **y** value must also be read. These values are read from a 'slope lookup list' inside of the *POLYGON* object. Because there are two sets of slope values, there are two 'slope lookup lists' inside every *POLYGON* object.

This *POLYGON* object specific information is stored in COMRAM at the offset address specified by the object list, and is aligned on DWORD boundaries. POLYGON objects are of variable length, and are composed of two types of DWORDs:

- Two start/initializing DWORD (used to initialize **x1**, **x2**, scanline)

- Any number of slope intercept DWORDs (storing values **s** and **y**)

Start DWORDs:
There are two start DWORDs, which can be found in COMRAM at the offset specified by the object list. These 8 bytes are used by the video processor to initialize the polygon rasterization process. The format of the first two DWORDs is the following:

dword 0	dword 1
?????aaaaaaaaaauuuuuuuuuuuuuuu	**vvvvvvvvvvvvvvvvyyyyyyyyy?cccccc**

a offset address of second 'slope lookup list'
u 16-bit value representing initial **x1** value (first pixel in scanline to be turned on)
v 16-bit value representing initial **x2** value (last pixel in scanline to be turned on)
y 9-bit value representing first scanline that polygon appears on
c 6-bit pixel color index value

It should be noted that the **x1** and **x2** values are 16-bit to allow for fractional pixel precision, which is necessary for retaining true positional information when adding slope information to the value.

Slope/intercept DWORDs:
There may be any number of slope intercept DWORDs in the *POLYGON* object. Each single DWORD represents one slope and scanline intercept pair. These values are strung together to make a 'slope lookup list', of which (as described above) there are two. The format of a slope intercept DWORD is the following:

dword
ssssssssssssssssyyyyyyyyy???????

s 16-bit value representing **s** value (slope value to be added to **x**)
y 9-bit value represents the last scanline that **s** is valid for

The value $FFFFFFFF, which is reserved, denotes not only the end of the 'slope lookup list', but the end of the *POLYGON* object as well. Note that this value is only contained in the first 'slope lookup list'. There is no value which signifies the end of the second 'slope lookup list'.

As described in a previous section, there are two 'slope lookup lists'. The first list (which operates on **x1**) begins where the two start DWORDs end (offset specified by object list + 4). The second list (which operates on **x2**) begins at the offset specified in the first *POLGYON* object start DWORD.

The video processor rasterizes a *POLYGON* object in the following manner.

```
void RasterizePOLYGON( WORD * COMRAM, int address1 )
{
      WORD x1, x2, s1, s2;
      int address2, y, y1, y2, color;

      address2 = COMRAM[address1++] & 0x08FF;
      x1 = COMRAM[address1++];
      x2 = COMRAM[address1++];
      y = COMRAM[address1] >> 7;
      color = COMRAM[address1++] & 0x003F;

      SetColor( ColorTable[color] );

      address1 = GetSlopeData( COMRAM, address1, &s1, &y1 );
      address2 = GetSlopeData( COMRAM, address2, &s2, &y2 );
      while(s1 != 0xFFFF)
      {
            DrawLine( x1>>7, y, x2>>7, y );
            x1 += s1;
            x2 += s2;
            y++;
            if (y >= y1)
                  address1 = GetSlopeData( COMRAM, address1, &s1, &y1 );
            if (y >= y2)
                  address2 = GetSlopeData( COMRAM, address2, &s2, &y2 );
      }
}

int GetSlopeData( WORD * COMRAM, int address, WORD * s, int * y)
{
      *s = COMRAM[address++];
      *y = COMRAM[address++] >> 7;
      return address;
}
```

Alphanumerics processor

TBD

RGB signal generator

TBD