PicomiteVGA

2022 programming challenge

SSTV.BAS

Experiments with SSTV

Introduction

SSTV (Slow Scan TeleVision) is a transmission standard for sending video pictures over an audio channel. Developed by amateur radio enthousiasts. Earliest eye-catching application is by NASA sending footage of the dark side of the moon. A company that has influenced SSTV in the early days was Robot. Since the early 1970's they deliver encoders and decoders, and have contributed to the standard. The early equipment consisted of analog electronics and picture tubes with storage capability.

Picture resolutions start at 120x120 pixels (Robot-8 format in B/W)

With the rise of the PC, and the audio cards, a new era for SSTV started. Digital signal processing, possibility for colour transmissions, and higher resolution pictures. Because the basics of the transmission standard where not changed, the higher picture quality lead to longer transmission times.

Modern standards are Martin(1/2) and Scottie(1/2/3) with typical resolutions of 320x240. The abundancy of standards also required a means to distinguish between them. Otherwise it is undoable to listen to you amateur radio channel, and actually create a SSTV picture before it disappears. For this reason the VIS (video information section) is send before the actual picture is transmitted.

SSTV highlights

There is significant amount of information on the web about SSTV, although not all websites are actual. Wikipedia shows basic information, and provides links to test streams.

SSTV uses audio tones in the spectrum between 1100Hz and 2300Hz. The picture transmission uses 1200Hz to indicate a SYNC, and 1500Hz...2300Hz to transmits various greyscale levels between black (1500hz) and white (2300Hz).

An SSTV picture starts with a vertical sync (vsync) with a duration of more than 20ms of 1200Hz. Then sequenctly video lines are transmitted (non interlaced), each line starting with a horizontal sync (5ms of 1200Hz).

In case color pictures are broadcast, each line consists of one horizontal sync, and then a sequence of all B pixels, then all G pixels, then all R pixels. Accurate timing allows remapping of the RGB data to coloured pixels.

Retrieving the video data from the audio stream has some complications.

The pixels are broadcast at 0.572ms per pixel. If you calculate that the duration of a single 1500Hz sine wave is 0.66ms, it should be clear that broadcast of a single pixel is not even presented by one whole sine wave (unlike early computer cassette storage standards). As can be seen when running the SSTV software, single pixel black picture area's are prone to errors.

An additional complication lies in the fact that the horizontal sync (5ms @1200Hz) gates the start of the pixel sampling for that line. With a period of 0.83ms the sync represents 1.5 pixel jitter.

Many of these complications can be overcome with digital filtering and sync re-alignment and acurate pixel acquisition. However … this is a Picomite Programming Challenge.

Picomite SSTV

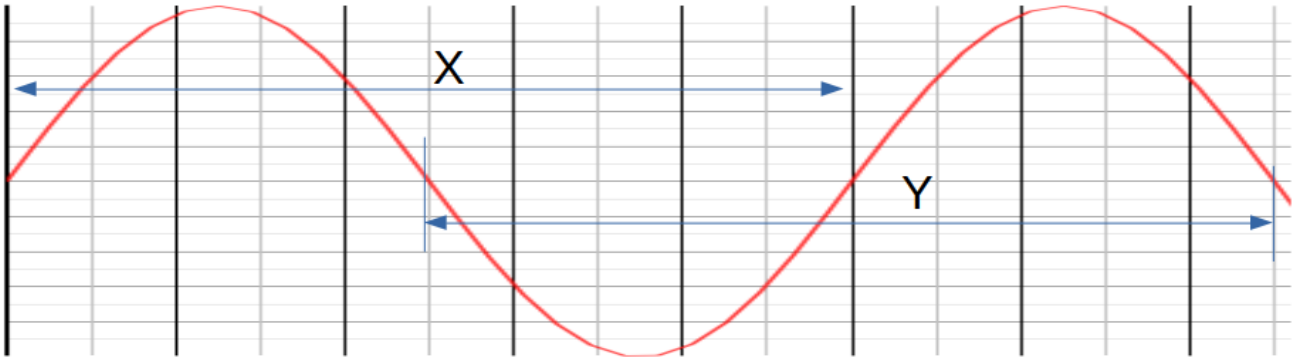
The PicoMiteVGA is a MMbasic computer created by Geoff Graham and Peter Mather, based on a Raspberry Pi pico (RP2040) board. The firmware is **5.07.04** as requested by the judges. Specific for this project there are some bug fixes in the current 5.07.05b5, but for the challenge these bugs have workarounds by using peeks and pokes (i.e. PIO READ is done through PEEK(WORD address))

The computer features 640x480 monochrome VGA or 320x240 16 colour VGA. It has a relatively fast Basic Interpreter running on a 126/252MHz ARM core.

One of the specific features of this chip is that it has 2 PIO sequencers with 32 steps program memory. These can be used to do fast real-time tasks independent of the ARM processor. The PicomiteVGA uses 1 PIO sequencer (PIO0) to generate VGA video.

Picomite SSTV: the PIO sequencer

The other PIO sequencer (PIO1) is used in this application to demodulate the audio tones (measure timing) at Picomite pin 1 (GP0). To accommodate a pixel rate higher than the sine wave period, each sine wave is samples twice (in- and out of phase) and timing send to a fifo, where the MMBasic program can pull it when needed.



The sine wave is measured using the PIO X register, 180 degrees out of phase the sine waves are measured using the PIO Y register. Zero crossings are used as a reference. By alternating the measurements an update is present every half sine wave (0.42ms @ 1200Hz), which is faster than the pixel clock.

Picomite SSTV: the basic program

The SSTV program as contributed to the programming challenge presents a chronological set of experiments to decode Robot-8 SSTV pictures (120x120 pixel picture in 256 grey scales). The experiments present my personal evolution in knowledge around SSTV and the PIO sequencer.

Note: the MMBasic program must be run at 252MHz by setting the option for CPU speed.

```
OPTION CPUSPEED 252000
```

When starting the program you are welcomed with a selection menu. Each entry will be explained here.

1/ frequency counter

The frequency counter uses the PIO to measure the period of the signal at Picomite pin 1 (GP0) and convert that to a frequency. This tool was used to make sure the PIO works.

2/ waterfall

The waterfall is a vertical scrolling graph showing samples taken from the PIO sequencer. The horizontal axis is the frequency. A frequency modulated signal (SSTV) shows like water falling down in regular patterns. The significant frequencies for SSTV are indicated in the top row.

3/ brute force sampling of picture

A vertical sync is used to start the sampling of the picture. From vertical sync the timing is determined by the free running MMBasic program. No synchronization. Timing is tuned by a single “pause” command and must be tuned very accurately to achieve a picture. Shown in mode 1 (640x480) as a small (120x120 pixels) stamp size picture in black and white.

4/ syncronized video capture

Both vertical sync and horizontal sync are detected, and pixel sampling is aligned to horizontal sync. A single “pause” command per sample is used to get roughly 120 pixels sampled. From this mode it can be seen that sync jitter (no digital filters) causes line offsets (right hand side of picture). Single black pixels in a white background and vise versa present an additional problem (sometimes they are missed).

5/ greenscale video

The Picomite features 16 colour RGB. The green colour has 2 bits per pixel. This allows for 4 level grey scale (green scale) video. This mode uses video mode 2, and exploits 4 brightness levels in green for representing the picture. Since the resolution in mode 2 is 320x240 the picture looks bigger.

6/ dithered greyscale

Few decades ago, newspapers where printed using black ink dithering to achieve different levels of grey. Dithering is a method to divide a pixel into a group of 4, 9 or 16 smaller pixels, and highlight (print) a number of these sub pixels depending on the brightness of the pixel. When seen from a distance the sub pixels tend to blur to a solid surface (similar to what a retina display on a mobile phone does). SSTV dithering scales each pixel to 3x3 sub pixels in mode 1. This gives the option to light between 0 and 9 sub pixels per pixel, representing 10 level greyscale. The 120x120 picture is represented as 360x360 sub pixels in VGA mode 1. When running the test stream this mode is a degradation for sharp text, but is exciting for pictures. This is no-where as good as 256 level grey scale, but the best that can be achieved using MMBasic on a monochrome mode. It is impressive to see that the basic interpreters can calculate and paint 9 pixels every 0.5ms.

The test stream

During development of the SSTV software a Robot-8 test stream was used from the Wikipedia page. This stream shows a few pages text in high contrast, and some pictures of girls faces in 256 level greyscale.

Wiki link: https://en.wikipedia.org/wiki/Slow-scan_television

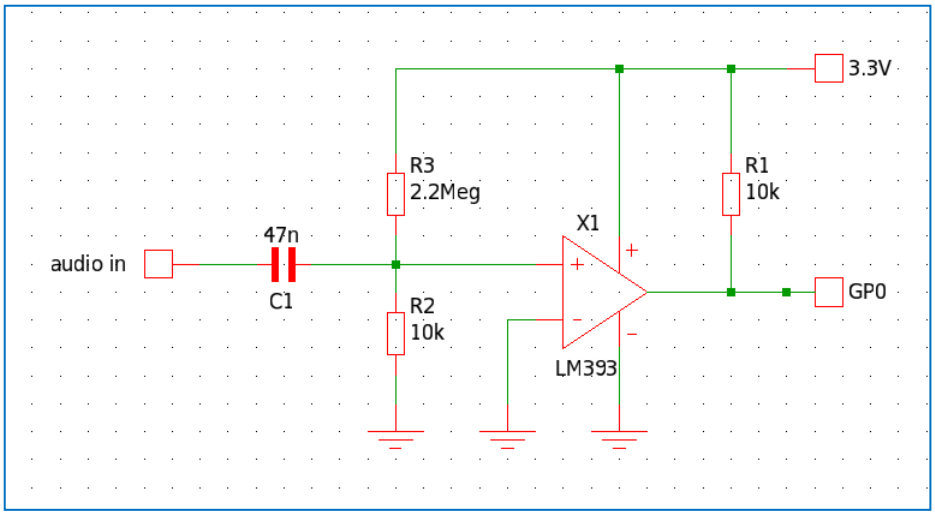
Stream link: https://www.youtube.com/watch?v=u3k6Xt30Z7g

Demo movie: picomite sstv - YouTube

In the demo, the stream is played from a laptop, where the audio (headphone output of laptop) is connected to the Picomite through a simple hardware interface (an amplifier) to pin 1 (GP0).

Hardware interface

The hardware interface is a single comparator that turns the input sine wave into a square wave. The 2.2meg resistor creates a little offset, so the comparator offset eliminates noise.



Further enhancements

The current picture decoding works around PAUSE statements. Original idea for sampling was using a SETTICK command at 0.572ms. However the SETTICK could not be programmed to fractions of a milisecond. A test was run using a hardware PWM running at 0.572ms (pixel clock) and reading the picomite GPIO pin that was toggled by the PWM. That test was successful, but did not give a significant improvement over the PAUSE implementation for Robot-8 SSTV, so it was dropped for the challenge.

The greyscale video decoding might be possible by BLIT-ing pre-defined 3x3 tiles (pixels). May be more efficient than drawing 9 individual pixels. Especially when the 10 different 3x3 tiles would be pre-calculated.

In case the accurate clocking would work, a simple SSTV Martin-1 colour decode could be tried. This would be a proof of concept, but would not satisfy. A 24bit colour RGB picture represented in 8(16) colours is rarely distinguishable at all.

In essence the video limitations of the PicomiteVGA hinder further development, causing development to stop here, and present the code as is.

Appendix A: the program

Below is a listing of the SSTV program as presented for the programming challenge 2022.

```
'SSTV decoder program for ROBOT-8 mode for PICOMITE MMBasic
Option default integer
MODE 1

'pio program measure overlapping rising and falling period and send to FIFO
'loops increment both X and Y, alternating X and Y are reset and pushed.
'0 E020   'set X=0
'1 A029   'X -> fffffff
'2 00C5   'jmp (pin=1) to loop2
'3 0084   'Y-- (first loop Y contains rubbish)
'4 0042   'count X-- loop1
'5 A0CA   'mov -Y to ISR
'6 8000   'push noblock
'7 E040   'set Y=0
'8 A04A   'Y -> fffffff
'9 004A   'count X-- loop2
'A 008B   'Y--
'B 00C9   'jmp (pin=1) in loop2
'C A0C9   'mov -X to ISR
'D 8000   'push noblock
'E 0000   'jmp 0 (rest is filled with 0 = jmp->0)
Dim a%(7)=(&h008400C5A029E020,&hE0408000A0CA0042,&h00C9008B004AA04A,&h8000A0C9,
0,0,0,0)
f=3e6     '3MHz @ 3cycles per loop = 1us per tick

'configure pio1
e=Pio(execctrl 0,0,&h1f)  'use gp0 for PIN
s=Peek(word &h503000d0)   'use old value
p=0                       'no GPxx pins for PIO1

'program pio1 and start
PIO program 1,a%()
PIO init machine 1,0,f,p,e,s
PIO start 1,0

'pio fifo register
ff=&h50300020

'sstv specific frequecies converted to time in us
l=1e6/1100 'sync low
r=1e6/1200 'sync
u=1e6/1300 'sync high
b=1e6/1500 'black
v=1e6/1900 'black/white threshold
w=1e6/2300 'white

menu:
CLS
Print "SSTV Robot-8 playground":Print
Print "1 = measure frequency"
Print "2 = waterfall"
Print "3 = picture in burst mode (no hsync)"
```

```
Print "4 = picture monochrome"
Print "5 = picture in 4 green levels"
Print "6 = picture in dithered white"
Print "7 = stop"
Do
 kn$=Inkey$
Loop Until kn$<>""
Select Case Val(kn$)
 Case 1
   GoSub frequency
 Case 2
   GoSub waterfall
 Case 5
   GoSub greenvideo
 Case 3
   GoSub videobrute
 Case 6
   GoSub greyvideo
 Case 4
   GoSub video
 Case 7
   End
End Select
GoTo menu



frequency:
'measure time and convert to frequency
Do
 cnt=Peek(word ff) 'read fifo pio 1 seq 0
 period = cnt+3      'period
 Print @(300,240) Int(1e6/period);" Hz      "
 Pause 100
Loop While Inkey$=""
Return



waterfall:
'show frequency graph on horizontal axis, scolling down
CLS
Do
 Text 1,1,"frequency (Hz)","LT"
 Text r/2,1,"1200","CT"
 Text b/2,1,"1500","CT"
 Text w/2,1,"2300","CT"
 Line u/2,20,u/2,479
 Line l/2,20,l/2,479
 For y=20 To 479
   cnt=Peek(word ff)
   Pixel cnt/2,y,1
   Pause 0.5   'delay to slow waterfall
 Next y
 CLS
Loop While Inkey$=""
```

```
Return



video:
'video decode using both hsync and vsync
're-syncing every line and frame
Do
 'CLS
 y=180
 Do
   x=240
   Timer =0
   'pixel video read an display
   Do
     Pause 0.3 'horizontal timing
     x=Min(390,x+1)
     p=Peek(word ff)
     Pixel x,y,(p<v)
   Loop Until p>u And Timer>58
   Timer =0
   'hsync detect > 3.5ms, vsync > 10ms
   Do
     Pause 0.3 'lower misses syncs
     p=Peek(word ff)
   Loop Until p<u And Timer>3.5
   If Timer > 10 Then Exit
   Inc y
 Loop
Loop While Inkey$=""
Return



videobrute:
'video decode fixed timing from vertical sync

brute:
'find vsync
Do
 Do
   p=Peek(word ff)
 Loop Until p>u
 Timer =0
 Do
   p=Peek(word ff)
 Loop Until p<b
Loop Until Timer > 10

'brute force process all pixels and hsync equally
For y=180 To 298
 For x=240 To 359
   p=Peek(word ff)
   Pixel x,y,(p<v)
   Pause 0.487
 Next x
```

```
Next y
If Inkey$="" Then GoTo brute
Return



greenvideo:
'video decode in mode 2 using 4 level green
're-syncing every line and frame
MODE 2
Dim c%(3)=(0,&h4000,&h8000,&hC000) '3 green levels and black

Do
 CLS
 y=50
 Do
   x=80
   'capture pixel data and display
   Timer =0
   Do
     Pause 0.3 'rough horizontal timing
     x=Min(220,x+1)
     p=Peek(word ff)
     i=(p<580)+(p<522)+(p<470)
     Pixel x,y,c%(i)
   Loop Until p>u And Timer>60
   Timer =0
   'detect end of hsync to start new line
   Do
     p=Peek(word ff)
   Loop Until p<b And Timer>4
   If Timer > 10 Then Exit 'sync is vsync
   Inc y
 Loop
Loop While Inkey$=""
MODE 1
Return



greyvideo:
'create greyscale out of 3x3 tiles in B/W pixels
're-syncing every line and frame

CLS
Do
 y=50
 Do
   x=100
   'dither pixels by drawing them individually depending grey level
   Timer =0
   Do
     Pause 0.1
     x=Min(500,x+1)
     k=y+1:l=k+1
     p=Peek(word ff)
     Pixel x,y,(p<571)
```

```
      Pixel x,k,(p<522)
      Pixel x,l,(p<480)
      Inc x
      Pixel x,y,(p<500)
      Pixel x,k,(p<632)
      Pixel x,l,(p<445)
      Inc x
      Pixel x,y,(p<600)
      Pixel x,k,(p<462)
      Pixel x,l,(p<545)
    Loop Until p>u And Timer>60
    Timer =0
    Print @(0,0) x
    'detect end of hsync to start new line
    Do
      p=Peek(word ff)
    Loop Until p<u And Timer>3.5
    If Timer > 10 Then Exit 'sync is a vsync
    Inc y,3
  Loop Until y>407
Loop While Inkey$=""
Return
```

Appendix B: The PIO sequencer explained.

Below is the sequencer program that is the heart of the SSTV program. I will try to explain why things are what they are.

```
'0 E020   'set X=0
'1 A029   'X -> fffffff
'2 00C5   'jmp (pin=1) to loop2
'3 0084   'Y-- (first loop Y contains rubbish)
'4 0042   'count X-- loop1
'5 A0CA   'mov -Y to ISR
'6 8000   'push noblock
'7 E040   'set Y=0
'8 A04A   'Y -> fffffff
'9 004A   'count X-- loop2
'A 008B   'Y--
'B 00C9   'jmp (pin=1) in loop2
'C A0C9   'mov -X to ISR
'D 8000   'push noblock
'E 0000   'jmp 0 (rest is filled with 0 = jmp->0)
```

First of all, the PIO sequencer cannot do a calculation. It can not add, it can not subtract. The only "arithmetic" it know is a (post) decrement of a value. So there is not even an increment ... only decrement.

The PIO has 2 registers X and Y (appart from the shift registers).

Normally the duration of an event can be measured as follows

```
counter = 0
DO
 INC counter
LOOP UNTIL event
```

But since there is no increment in the PIO it has to be written as

```
counter = -1
DO
 DEC counter
LOOP UNTIL event
counter = -counter
```

The preset value of -1 is because the PIO cannot do a arithmetic counter = -counter, but it can invert bits. And when you invert -1 (&hffffffff) you get 0.

Above is exactly what the PIO program does, only the instructions to do this are a bit non-intuitive.

The "event" in above code is the state change of a GPIO pin of the pico. The most logical method the PIO has to do this is to use a conditional jump instruction (JMP). In this instruction a pre-defined pin can be sampled. This pre-defined pin is defined in the EXECCTRL register (not in the PINCTRL register) and is completely independent of pins used for SET, OUT, IN, SIDE-SET. And the conditional JMP can only jump when the pin is HIGH. So there is no JMP on pin low. This restriction determines the flow of the program.

```
DO
 X=-1
loop1:
 JMP on PIN to loop2
 DEC Y
 DEC X and JMP to loop1
loop2:
```

```
 PUSH Y
 Y=-1
loop3:
 DEC Y
 DEC X
 JMP on PIN to loop3:
 PUSH X
LOOP
```

So the input pin condition is tested in loop1, where the condition escapes the counting loop. And the input pin condition is tested in loop2(3) where it continues the counting loop.

Now it gets a bit confusing. The DEC instruction in above pseudo code does not exist in the PIO instruction set. It is a side effect (post decrement) of the JMP instruction. DEC in fact is:

```
 JMP (Y--) next_instruction
next_instruction:
```

So regardless the initial value of Y, JMP either goes to the next instruction, or it jumps to the next instruction. But it has decremented Y.

How do we get the data from the counter loops. In above code we see a PUSH pseudo instruction. The PIO can push data to a FIFO that can be ready by the ARM processor. However , it can only push data into the FIFO from the ISR (Input Shift Register). So the PUSH pseudo instruction is implemented as follows:

```
 MOV X to ISR and invert all bits (counter = -counter)
 PUSH ISR to FIFO
```