

Pico Video Game Driver for SX/B 2.0

The “Pico Game Driver” is an interrupt driven sound and monochrome video driver. By using interrupts to generate the video and sound, you can focus on the details of your game without having to worry about timing.

Sound

The sound portion of the driver is very simple. You simply call the subroutine named SND that takes three parameters, Frequency, Duration, and Volume. Note that the tone is produced by the interrupt driver, so your program does not stop while the sound is being produced. If you use two SND commands without a delay between them you will not hear the first SND tone because the program will immediately execute the second SND command after the first.

Frequency (1 – 255) : This parameter specifies the frequency of the sound produced. The frequency is $15723 / \text{value}$. If you specify a frequency of 1 the tone will be 15723 Hz. If you specify a value of 255 the frequency of the tone will be $(15723 / 255)$ or 61.66 Hz.

Duration (0 – 255) : This parameter specifies how many video frames (1/60 second) the tone will continue. A duration of 60 would last for 1 second. If duration is zero, then the tone continues forever or until another sound command is issued. You can detect when a sound is finished by checking the driver variable sndDurCnt. When sndDurCnt is equal to zero, the sound is finished.

Volume (0 – 15) : This parameter specifies the volume of the tone.

Video

The screen is made up of 176 cells. There are 16 cells horizontally and 11 cells vertically. Each cell can display one of sixteen tiles from a tile set. Each of the 16 tiles consists of an 8x8 monochrome bitmap image.

You can configure the driver to use either one or two tile sets. With one tile set, all of the 11 horizontal lines displays tiles from the same tile set. If you configure for two tile sets, then the top line displays tiles from the first tile set, and all other lines displays tiles from the second tile set. Using one tile set will free up 128 program instructions allowing you to create longer more complicated programs.

By default the driver uses two tile sets. To configure the driver for one tile set you must define the word “TILESETS_1” by using:

```
{ $DEFINE TILESETS_1 }
```

The reason for giving the top line it's own tile set is so you can display values (score, timer, etc) which will require 10 tiles (for digits 0 thru 9).

The driver includes two routines to increment (ValueInc) and decrement (ValueDec) values. These routines require that the digits 0 thru 9 be stored in tiles 0 thru 9. Therefore it is advisable to use tile15 for the blank tile.

Here is the driver memory map when using one tile set:

\$000 - \$1FF = Game Subroutines and Pico Game Driver Code
\$200 - \$27F = Tile Set 1 (Line 0-10 tile set bitmaps)
\$280 - \$7FE = Game program and Pico Game Driver subroutines

Here is the driver memory map when using two tile sets:

\$000 - \$1FF = Game Subroutines and Pico Game Driver Code
\$200 - \$27F = Tile Set 1 (Line 0 tile set bitmaps)
\$280 - \$2FF = Tile Set 2 (Lines 1-10 tile set bitmaps)
\$300 - \$7FE = Game program and Pico Game Driver subroutines

Screen locations can be specified either by “position” which is a value from 0 (top left) to 175 (bottom right). The screen positions start in the top left with 0, and increase as you go across the line left to right. So the position value of the top right cell is 15, then the second line is positions 16 through 31 and so on down the screen. Or you can simply specify the X and Y screen position. X is horizontal from 0(left) to 15(right), and Y is vertical from 0(top) to 10(bottom).

The entire screen is stored in an array named “videoScreen” that has 88 elements. Two cells are stored in each element of the array. The most significant nibble is the cell on the left, the least significant nibble is the cell on the right.

If you wanted to put tile 0 at position 0, and tile 1 at position 1 you could do:

```
PutTile 0, 0  
PutTile 1, 1
```

or you could simply do:

```
videoScreen(0) = $01
```

Also each line is aliased as an array videoLine0 through videoLine10. These arrays contain 8 elements(bytes) that hold the tile values for that particular line on the screen.

Here are the I/O pins that are defined by the driver

LEDs	PIN RA OUTPUT	
JStkUp	PIN RB.0 INPUT PULLUP	' Joystick input pins
JStkDown	PIN RB.1 INPUT PULLUP	' Joystick input pins
JStkLeft	PIN RB.2 INPUT PULLUP	' Joystick input pins
JStkRight	PIN RB.3 INPUT PULLUP	' Joystick input pins
JStkFire	PIN RB.4 INPUT PULLUP	' Joystick input pins
IO_0	PIN RB.5 INPUT PULLUP	
IO_1	PIN RB.6 INPUT PULLUP	
IO_2	PIN RB.7 INPUT PULLUP	
AVPort	PIN RC OUTPUT	' Connects to Audio/Video R2R DACs
AVBlack	PIN RC.2 OUTPUT	' Make high for black output
AVWhite	PIN RC.3 OUTPUT	' Make high (along with black) for white output

Here are the constants that are defined by the driver

Pressed	CON 0	' Buttons go to zero when pressed
VidMode_VSync	CON 0	' Video modes: 0 = Generating Vertical Sync
VidMode_TBlank	CON 1	' 1 = Generating Top Blank Lines
VidMode_Active	CON 2	' 2 = Generating Active Video Lines
VidMode_BBlank	CON 3	' 3 = Generating Bottom Blank Lines
Line0	CON 0	' Offset for video line 0
Line1	CON 16	' Offset for video line 1
Line2	CON 32	' Offset for video line 2
Line3	CON 48	' Offset for video line 3
Line4	CON 64	' Offset for video line 4
Line5	CON 80	' Offset for video line 5
Line6	CON 96	' Offset for video line 6
Line7	CON 112	' Offset for video line 7
Line8	CON 128	' Offset for video line 8
Line9	CON 144	' Offset for video line 9
Line10	CON 160	' Offset for video line 10

Here are the variables that are defined by the driver

videoScreen	VAR BYTE(88) SPAN	' Whole screen lines 0 through 10
videoLine0	VAR BYTE(8) @ \$30	' 1st line of text (line 0)
videoLine1	VAR BYTE(8) @ \$38	' 2st line of text (line 1)
videoLine2	VAR BYTE(8) @ \$50	' 3rd line of text (line 2)
videoLine3	VAR BYTE(8) @ \$58	' 4th line of text (line 3)
videoLine4	VAR BYTE(8) @ \$70	' 5th line of text (line 4)
videoLine5	VAR BYTE(8) @ \$78	' 6th line of text (line 5)
videoLine6	VAR BYTE(8) @ \$90	' 7th line of text (line 6)
videoLine7	VAR BYTE(8) @ \$98	' 8th line of text (line 7)
videoLine8	VAR BYTE(8) @ \$B0	' 9th line of text (line 8)
videoLine9	VAR BYTE(8) @ \$B8	' 10th line of text (line 9)
videoLine10	VAR BYTE(8) @ \$D0	' 11th line of text (line 10)
vidCnt	VAR BYTE (1)	' vsync=6, topblank=40, active=240, botblank=24
vidMode	VAR BYTE (1)	' 0=vsync, 1=topblank, 2=active, 3=botblank
vidLineRpt	VAR BYTE (1)	' Repeat each line counter
vidTileLine	VAR BYTE	' Current line within 1 tile (cannot be an array)
tileAddr	VAR BYTE	' RAM address of tile (cannot be an array)
tileDots	VAR BYTE	' Tile line dot pattern (cannot be an array)
sndDurCnt	VAR BYTE (1)	' Counts frames for sound duration
sndToneCnt	VAR BYTE (1)	' Counts video lines for sound
sndTone	VAR BYTE (1)	' Current sound tone value
sndVol	VAR BYTE (1)	' Volume of current sound
intFlag	VAR BIT	' Cleared by video interrupt

Here are the subroutines that are included in the driver

Snd generates tones

Snd Freq(1-255), Duration(0-255), Volume(0-15)

PutTile is used to place tiles on the screen.

PutTile Position, TileID

PutTile X, Y, TileID

GetTile is used to retrieve the TileID at a certain screen location.

var = **GetTile** Position, TileID

var = **GetTile** X, Y, TileID

If var is a BYTE it will be set to the TileID the position, if var is a WORD, the LSB will be set to the TileID, and the MSB will be set to the RAM location of the position specified.

Delay is used to pause or slow down the program.

Delay value

Delays for about "value" milliseconds.

WaitSync is used to wait until the screen has been sent to the TV.

WaitSync

WaitSync is useful to avoid changing the screen while it is being sent to the TV.

ClearScreen is used to fill all or part of the screen with a specified tile.

ClearScreen TileID

ClearScreen TileID, StartLine

If "StartLine" is specified, only lines from that line down are filled with "TileID"

ScrollUp is used to move the screen up one line.

ScrollUp

ScrollUp TileID

If "TileID" is given, the bottom line is filled with this tileID, otherwise the bottom line is left intact.

ScrollDown is used to move the screen down one line.

ScrollDown

ScrollDown TileID

If "TileID" is given, the top line is filled with this tileID, otherwise the top line is left intact.

ScrollLeft is used to move the screen left one column.

ScrollLeft TileID

The right column is filled with tileID.

ScrollRight is used to move the screen right one column.

ScrollRight TileID

The left column is filled with tileID.

ValueInc is used to increase a value that is displayed on the screen. A score for example.

ValueInc Position

ValueInc X, Y

The position given must be the location of the digit in the value that you wish to increment. The digit MUST match the TileID (TileID 0 must be the bitmap of zero, etc). When the digit specified reaches “9”, it is made a “0” and the previous position (if it’s tileID is 0 to 9) is incremented.

ValueDec is used to decrease a value that is displayed on the screen. A timer for example.

ValueDec Position

ValueDec X, Y

The position given must be the location of the digit in the value that you wish to decrement. The digit MUST match the TileID (TileID 0 must be the bitmap of zero, etc). When the digit specified reaches “0”, it is made a “9” and the previous position (if it’s tileID is 0 to 9) is decremented.

PlotXY is used to plot pixels when using the pixel tile sets. The pixel tile sets use all 16 tiles with various quarters of the tiles filled. These tile sets are loaded by defining LOAD_PIXEL_TILESET_1 and/or LOAD_PIXEL_TILESET_2 before loading “PicoNTSC_INC.SXB’.

PlotXY PixelX(0-31), PixelY(0-21)

Plot makes the pixel black.

PlotXY is used to plot pixels when using the pixel tile sets. The pixel tile sets use all 16 tiles with various quarters of the tiles filled. These tile sets are loaded by defining LOAD_PIXEL_TILESET_1 and/or LOAD_PIXEL_TILESET_2 before loading “PicoNTSC_INC.SXB’.

UnPlotXY PixelX(0-31), PixelY(0-21)

UnPlotXY makes the pixel white.

There are a number of conditional defines that are used to configure the driver

```
`{$DEFINE LOAD_PIXEL_TILESET_1}
```

This will load the bitmaps for tile set 1 that work with the PlotXY and UnPlotXY commands.

```
`{$DEFINE LOAD_PIXEL_TILESET_2}
```

This will load the bitmaps for tile set 2 that work with the PlotXY and UnPlotXY commands.

```
`{$DEFINE TILESETS_1}
```

This will use tile set 1 for ALL video lines. This frees 128 program instructions for your game.

NOTE: This must be defined BEFORE PicoNTSC_DEF.SXB is loaded.

The best way see how the driver works is by examining the demo programs that are provided.

BrickOut.SXB is a simple breakout type game

Drive Game.SXB is a simple car “stay on the road” type game (scrolls down)

Flip-It.SXB is a simple board game

Pong.SXB is a simple pong game

Sketch.SXB is a simple drawing game

SX Invaders.SXB is a simple space invaders type game

Template.SXB is a bare template program (doesn't do anything)

The “Pico Video Game Driver” was written by:

Terry Hitt

Hitt Consulting

www.HittConsulting.com

terry@hittconsulting.com

Comments and suggestions are welcome.