Video Graphics Using the SPI on the MC68HC11 Microcontroller

Christopher R. Carroll

Electrical and Computer Engineering
University of Minnesota Duluth

Abstract

The Serial Peripheral Interface (SPI) input/output capability of the MC68HC11 microcontroller is a feature of the MC68HC11 architecture that is often overlooked by casual experimenters. It is designed to interface to input/output devices that include special hardware specifically meant to connect to the SPI. However, the SPI provides a handy way to output a generic high-speed stream of bits from the MC68HC11 without requiring additional external hardware. That capability has been employed to generate the video signal for an alphanumeric text display on a standard video monitor, as described in an earlier ASEE paper[1].

This paper details a technique for producing a simple graphics display on a standard video monitor, using the SPI unit to generate the high-speed bit stream necessary for the video signal driving the monitor. The display produced is adequate for simple line graphs or other comparable displays. The heart of the technique described in this paper is controlling the timing of data emerging from the SPI very carefully, at the clock cycle level, and thus establishing the position of various graphical elements along the scanlines of the standard video display. The technique relies heavily on creative programming techniques to achieve this clock-cycle-level control of the signal timing, clearly demonstrating the operation of the SPI unit while at the same time serving as a useful graphics output utility that can be used by other software.

The software routines that control the MC68HC11's SPI unit to produce the graphics output are revealed in this paper, as are the few discrete components necessary to produce a composite video signal to drive a standard video monitor. Equipping an MC68HC11 microcontroller with this feature adds a handy output function that can be used in any MC68HC11 system.

Copyright ©2004, American Society for Engineering Education

Hardware Design

The video graphics system described in this paper is an innovative application of the Serial Peripheral Interface (SPI) on the MC68HC11 microcontroller[2], implemented with creative software that controls the SPI using detailed timing in the program. The video graphics produced by this design are adequate for simple line graphs or similar images. Figure 1 shows the display produced by the author's implementation. A warning – if you as the reader of this paper are not interested in low-level assembly language code and counting clock cycles of execution time on an instruction-by-instruction basis, you will have no interest in this paper. However, if designs that squeeze out the last measure of performance from a minimal amount of hardware interest you, as they do the author, then read on.

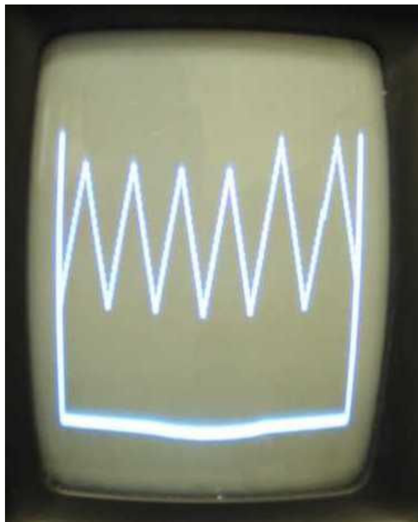

Figure 1. Typical graph display

The composite video signal[3] produced by this system drives a standard video monitor. The monitor is turned on its side so that scanlines run vertically from bottom to top. This is done so that the limited resolution along a scanline (about 50 pixels at the 1 Mbit/second video rate possible from the SPI) does not limit the application. The hardware involved in producing the composite video signal is shown in Figure 2. The circuit shown there combines a frame-sync signal from the MC68HC11's Port A bit 3, a scanline-sync signal from Port D bit 5, and a video signal from Port D bit 3 to generate the composite video signal. The frame-sync signal is produced by interrupts generated by the Output Compare 5 unit in the MC68HC11. The scanline-sync signal is generated by software in the service routine for that interrupt. The video signal is the "Master-Out-Slave-In" data output of the SPI. The inverters shown in Figure 2 come from a 74LS05 chip, and have open-collector outputs. They could easily be replaced with discrete transistors, but the 74LS05 was available in the system and provided a quick way to combine the three signals. The values of the resistors are chosen to produce acceptable sync levels and brightness on the particular video monitor driven by the composite video signal in the user's system.

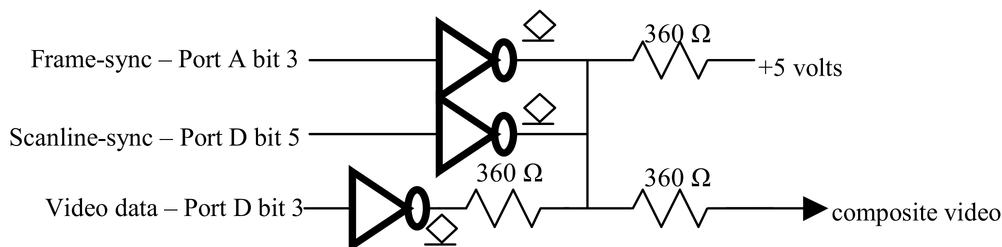

Figure 2. Composite video generation – resistor values chosen to adjust brightness and contrast.

This simple circuit is the only additional hardware required to implement the video display described here. All the interesting activity occurs in the interrupt service routine for the MC68HC11's Output Compare 5 unit.

Copyright ©2004, American Society for Engineering Education

Figure 2. Hardware required for composite video signal generation.
Note: Adjust resistor values to obtain desired brightness/contrast.

Software Design

Software that runs the video graphics system is based on a periodic interrupt generated by the Output Compare 5 hardware in the MC68HC11. The service routine for the interrupt generates 240 scanlines of image, each consuming exactly 63.5 microseconds, or 127 clock cycles of the MC68HC11 system running with an "E clock" of 2 MHz. The interrupt repeat period, in order to be an integer multiple of the scanline period, is 16.7005 milliseconds, or 33401 clock cycles. This period gives an interrupt rate of 59.88 Hz, close enough to the nominal 60 Hz frame refresh rate for standard video monitors. Port A bit 3, controlled by the Output Compare 5 interrupts, thus is used for the frame-sync signal. That port bit is driven low on each interrupt to begin a frame. The core of the interrupt service routine is shown below in Figure 3.

```
Graph:                ; output compare 5 interrupt entry, 59.88(10) Hz
       ldaa   #8                                                    - 2 clocks
       staa   1023    ; clear output compare 5 flag in TFLG1        - 4 clocks
       ldd    101e    ; output compare 5 event time                 - 5 clocks
       addd   #8279   ; timer increment (33401(10))to set frame rate - 4 clocks
       std    101e    ; establish next interrupt time               - 5 clocks
       ldaa   #13     ; # of dark scanlines to start frame          - 2 clocks

GLdark:               ; exactly 127(10) clock cycles per scanline
       bsr    GDark                                                 - 6 clocks
       deca                                                         - 2 clocks
       bne    GLdark                                                - 3 clocks

GLleft:               ; left side of graph - 127(10) clock cycles
       bsr    GLight                                                - 6 clocks
       brn    GLleft  ; wasted time                                 - 3 clocks
       ldaa   #c8     ; 200(10) scanlines of data                   - 2 clocks

GLdata:               ; data scanlines - 127(10) clock cycles each
       bsr    GData                                                 - 6 clocks
       deca                                                         - 2 clocks
       bne    GLdata                                                - 3 clocks

GLrght:               ; right side of graph - 127(10) clock cycles
       bsr    GLight                                                - 6 clocks
       brn    GLrght  ; wasted time                                 - 3 clocks
       ldaa   #13     ; # of dark scanlines at right side           - 2 clocks

GLend:                ; finish the frame - 127(10) clock cycles each
       bsr    GDark                                                 - 6 clocks
       deca                                                         - 2 clocks
       bne    GLend                                                 - 3 clocks

       inc    1020    ; output compare 5 now sets frame sync        - 6 clocks
       ldaa   #8                                                    - 2 clocks
       staa   100b    ; force the frame sync pulse                  - 4 clocks
       dec    1020    ; frame sync returns to 0 on next interrupt   - 6 clocks
       rti            ; return from the interrupt                   - 12 clocks
```

Figure 3. Interrupt Service Routine for Output Compare 5 (numbers in base 16)

As can be seen from a perusal of the code in Figure 3, the service routine starts with housekeeping to clear the interrupt flag and establish the next interrupt time. Then it generates 19 (=13_{16}) scanlines with nothing on them as a left "border" of the frame, one scanline with a lit "bar" marking the left side of the graph, 200 (=$c8_{16}$) scanlines showing the 200 points of graphed

Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition
Copyright ©2004, American Society for Engineering Education

data, one scanline with a lit "bar" marking the right side of the graph, and then 19 (=13_{16}) dark scanlines for the right "border" of the frame. Each of the three kinds of scanlines is generated by a separate subroutine, GDark, GLight, and GData, respectively, each of which executes in exactly 116 clock cycles, since scanlines must occur every 127 clock cycles during the frame, and there are 11 clock cycles worth of instruction execution between each scanline subroutine execution, as shown in Figure 3. Following the 240 scanlines, bit 3 of port A is forced high to produce the frame-sync pulse and prepare for the next interrupt.

```
GDark:              ; dark scanline - 116₁₀ clock cycles
       com    1008   ; scanline sync on  - 6 clock cycles
       ldab   #2     ; time delay        - 2 clock cycles
GWdark:
       decb                              - 2 clock cycles
       bne    GWdark                     - 3 clock cycles
       nop                               - 2 clock cycles
       nop                               - 2 clock cycles
       clr    1008   ; scanline sync off - 6 clock cycles
       ldab   #0f    ; time delay        - 2 clock cycles
GWdrk2:
       decb                              - 2 clock cycles
       bne    GWdrk2                     - 3 clock cycles
       nop                               - 2 clock cycles
       nop                               - 2 clock cycles
       nop                               - 2 clock cycles
       rts                               - 5 clock cycles
```

Figure 4. Subroutine to generate dark scanlines (numbers in base 16)

The GDark subroutine, shown in Figure 4, has just one job to do, namely generate the scanline-sync signal on Port D bit 5. Otherwise the subroutine just wastes time to consume its 116 clock cycles. It begins by complementing Port D, which turns on bit 5 and starts the scanline-sync pulse. Than, after 16 clock cycles of instruction execution, it clears Port D to complete the pulse. Everything else in the GDark subroutine is just wasting time. Nothing is put out on the video.

```
GLight:             ; light scanline - 116₁₀ clock cycles
       com    1008   ; scanline sync on  - 6 clock cycles
       ldab   #2     ; time delay        - 2 clock cycles
GWlght:
       decb                              - 2 clock cycles
       bne    GWlght                     - 3 clock cycles
       nop                               - 2 clock cycles
       nop                               - 2 clock cycles
       clr    1008   ; scanline sync off - 6 clock cycles
       ldab   #2     ; time delay        - 2 clock cycles
GWlgt:
       decb                              - 2 clock cycles
       bne    GWlgt                      - 3 clock cycles
       nop                               - 2 clock cycles
       ldab   #10                        - 2 clock cycles
       stab   1028   ; turn on video     - 4 clock cycles
       ldab   #0b    ; time delay        - 2 clock cycles
GWlgh:
       decb                              - 2 clock cycles
       bne    GWlgh                      - 3 clock cycles
       ldab   #50                        - 2 clock cycles
       stab   1028   ; turn off video    - 4 clock cycles
       rts                               - 5 clock cycles
```

Figure 5. Subroutine to generate light scanlines (numbers in base 16)

Copyright ©2004, American Society for Engineering Education

The GLight subroutine, shown in Figure 5 on the previous page, has the additional job of turning on the video for a portion of the scanline to form the left or right edge of the displayed graph. This subroutine still must generate the pulse on scanline-sync just like GDark does, with exactly the same timing. The code in Figure 5 shows how these two jobs are accomplished. The scanline-sync pulse is started by complementing Port D, and then completed by clearing Port D after wasting some time, just as in GDark. Then a little more time is wasted, and the SPI system is disabled. This has the effect of making the "Master-Out-Slave-In" signal low, which is used for the video signal, and that action lights the scanline on the screen. The scanline remains lit until the SPI is re-enabled after another time delay in the subroutine that determines how long the lit portion of the scanline lasts.

The third, and most complicated, subroutine that generates scanlines is GData, which produces scanlines that show one point of the displayed line graph. The code for GData is shown in Figure 6 on the next page. This code must generate the scanline-sync signal with the same timing as GDark and GLight do. It also generates a "baseline" to form the bottom of the displayed graph. Thirdly, it must light a pixel at a position along the scanline determined by the value of the data to be displayed at that position on the graph. The data to be displayed are contained in a table in memory of 200 bytes, pointed at by the Y index register in the MC68HC11. This GData subroutine must fetch the next data point out of the table and delay by a variable amount of time that depends on the data, before putting out the pixel representing the graphed point. Of course, the subroutine must execute in exactly 116 clock cycles regardless of the data value, just like GDark and GLight. This is a VERY tricky operation, and forms the meat of the results presented in this paper.

The SPI in the MC68HC11 used by the author emits bits at a maximum rate of 1 microsecond per bit, or one bit every two clock cycles of instruction execution. Thus, in order to adjust the position of a pixel along a scanline with a 1-pixel resolution, the time at which that pixel is emitted by the SPI must be adjusted with a resolution of 2 clock cycles. Achieving such a tight resolution on a software time delay is not easy. One cannot use a software loop that goes around a variable number of times, because the minimum loop time is more than 2 clock cycles and thus that technique cannot achieve the required timing resolution for delaying pixel generation. The technique used in GData is to use a string of "nop" instructions, each of which executes in two clock cycles, and to jump into that string of instructions at a position determined by the value of the data to be displayed on the graph at that point. By changing the number of "nop" instructions skipped in the software, the time at which the pixel is generated on the scanline is adjusted. Each "nop" that is skipped changes the position along the scanline at which the SPI emits the data pixel by two clock cycles, or one pixel position. Fewer "nop" instructions executed means that the pixel is emitted earlier (lower) on the scanline. Of course, since the total execution time for the GData subroutine must be fixed at 116 clock cycles, there must be a second set of "nop" instructions in which a compensating number are skipped to make the total number executed constant. This leads to a physically cumbersome subroutine, but an elegant one timing-wise that allows pixels in the line graph to be positioned along the scanline (vertically) with a resolution of two clock cycles, or one pixel.

Because the scanline time is limited, and the execution time of the GData subroutine is fixed, there is a limited range of data that can be plotted on the displayed graph using this technique. The implementaion shown in Figure 6 allows graphed data values between 0 and 18 (=12_{16}).

Copyright ©2004, American Society for Engineering Education

```
Gdata:              ; data scanline - 116_10 clock cycles, or 58 micro seconds
       com    1008  ; scanline sync on                         - 6 clock cycles
       psha         ; save value in accumulator A              - 3 clock cycles
       ldab   0,y   ; retrieve next data coordinate (0-18_10) - 5 clock cycles
       negb                                                    - 2 clock cycles
       ldaa   #fe   ; 0 in low bit is what turns on video      - 2 clock cycles
       bita   1029  ; just touch the SPI status register       - 4 clock cycles
       clr    1008  ; scanline sync off                        - 6 clock cycles
       staa   102a  ; SPI data - base line of graph            - 4 clock cycles
       ldx    #GPre ; pointer into code below                  - 3 clock cycles
       addb   #12   ; calculate jump offset used below         - 2 clock cycles
       abx                                                     - 3 clock cycles
       iny          ; move to next data point                  - 4 clock cycles
       jmp    0,x   ; jump into first block of nop's           - 3 clock cycles
GPre:
       nop          ; 18_10 nop's into which jump is indexed   - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       ldaa   #fe   ; pixel to turn on data point              - 2 clock cycles
       bita   1029  ; just touch the SPI status register       - 4 clock cycles
       staa   102a  ; store to SPI data                        - 4 clock cycles
       negb         ; calculate compensating index             - 2 clock cycles
       addb   #12                                              - 2 clock cycles
       lslb                                                    - 2 clock cycles
       abx          ; adjust x for compensating jump           - 3 clock cycles
       jmp    0f,x  ; jump into next string of nop's           - 3 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycles
       nop                                                     - 2 clock cycle
       pula         ; restore saved value of accumulator A     - 4 clock cycles
       nop          ; timing                                   - 2 clock cycles
       rts                                                     - 5 clock cycles
```

Figure 6. Subroutine to produce data pixels (numbers in base 16)

Copyright ©2004, American Society for Engineering Education

If any data value in the table of 200 data points is outside of the range 0-18, this software will crash. However, other code in the user's software can check and guarantee that values to be graphed are in the proper range.

Examining the code for GData, we see that the scanline-sync pulse is started in the same way as in GDark and GLight, by complementing Port D. Rather than waste time during the pulse, however, a few things are done during the 16 clock cycles before clearing Port D to end the scanline-sync pulse. These things include retrieving the next byte of data to be graphed from the table pointed at by index register Y, and preparing the pixel for output through the SPI to form the base line of the graph. Any 0's in the byte stored to the SPI for transmission result in lit pixels on the scanline, so the value of fe_{16} lights just one pixel when it is stored to the SPI data in the instruction following the clear of Port D to end the scanline-sync pulse. Note that the operation of the SPI requires reading the SPI status register at address 1029_{16} before storing data to be shifted out at address $102a_{16}$. Nothing needs to be done with the status byte, but it must be touched. That is the purpose of the "bita" instruction. Following the store to the SPI shift register, the value in index register X is calculated to point into the first string of "nop" instructions at the appropriate place, and then an indexed jump is performed, implementing a variable delay with a resolution of 2 clock cycles or 1 pixel time, the executing time for a "nop" instruction. Following that delay, whose value depends on the data to be graphed, a pixel is turned on by storing fe_{16} to the SPI shift register (after first touching the SPI status register, as before) to show the data point on the graph. Then the value in index register X is adjusted to implement a compensating delay by jumping into a second string of "nop" instructions so that the total execution time of the GData subroutine is constant at 116 clock cycles regardless of the value of the data point graphed on that scanline. Accumulator A is saved and restored in this subroutine because it's value is used to count data scanlines in the core part of the interrupt service routine, discussed earlier.

Summary

The combination of hardware and software presented here produces a usable graphics display on a standard composite video monitor suitable for displaying line graphs or similar simple graphic images. In order to use this package, the main program software must do several things, such as:

- establish the interrupt vector for Output Compare 5 to point to "Graph"
- turn on bits of DDRD to make appropriate port bits outputs
- set TCTL1 to clear Port A bit 5 on Output Compare 5 interrupts
- enable Output Compare 5 interrupts in TMSK1
- enable the SPI as a master in SPCR.
- enable interrupts with the "cli" instruction

These initialization steps can be included easily along with other initializations in the main program of whatever application uses the graphic capability described here. Data to be graphed must be in a table of 200 bytes in memory, and the main program must point index register Y to the beginning of that table. The data to be plotted must be in the range 0-18 ($0\text{-}12_{16}$). Although much fancier graphing capabilities are, of course, available, the system described here accomplishes a lot with very little, always the goal of any engineering endeavor. It should be easy to incorporate the graphing capability described here into almost any system using an MC68HC11 microcontroller.

Copyright ©2004, American Society for Engineering Education

References

1. Carroll, C. R., R. Alba-Flores, F. Rios-Gutierrez, "New Life for the MC68HC11 Evaluation Board," *2002 ASEE Annual Conference Proceedings*, Montreal, Canada (2002).

2. Spasov, Peter, *Microcontroller Technology: The 68HC11*, Fourth Edition, Prentice Hall, 2002.

3. Internet web page defining composite video signal, www.*rickard.gunee.com/projects/video/pic/howto.php*.

Biography

CHRISTOPHER R. CARROLL
Christopher R. Carroll received a Bachelor of Engineering Science from Georgia Tech, and M.S. and Ph.D. degrees from Caltech. After teaching in Electrical Engineering at Duke University, he is now Associate Professor and Assistant Head of Electrical and Computer Engineering at the University of Minnesota Duluth. His interests include special-purpose digital systems, VLSI, and microprocessor applications, especially in educational environments.

Copyright ©2004, American Society for Engineering Education