ARMY RESEARCH LABORATORY

Interfacing the TCM8230MD CMOS Camera With an ARM7 Microcontroller

by Justin L. Shumaker

NOTICES

Disclaimers

Army Research Laboratory

Aberdeen Proving Ground, MD 21005-5066

ARL-TN-344   **March 2009**

Interfacing the TCM8230MD CMOS Camera
With an ARM7 Microcontroller

Justin L. Shumaker
Vehicle Technology Directorate, ARL

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
March 2009	Final	May 2008–July 2008

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Interfacing the TCM8230MD CMOS Camera With an ARM7 Microcontroller	
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Justin L. Shumaker	1L162618AH80
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
U.S. Army Research Laboratory ATTN: AMSRD-ARL-VT-UV Aberdeen Proving Ground, MD 21005-5066	ARL-TN-344

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The TCM8230MD is a color complementary metal-oxide semiconductor video camera manufactured by Toshiba. Image resolutions range from 128×96 to 640×480 pixels, with frame rates as high as 30 Hz. Due to high data rates, a powerful microcontroller with an efficient buffering algorithm is required. Because of its small size, mass, power consumption, and cost, this camera has tremendous application to microrobotic systems. Microcontrollers such as the ARM9 have dedicated hardware to simplify interfacing this or any other camera with an inter-integrated circuit, 85-MHz, 8-bit data bus interface. However, the ARM9 is a more complicated microcontroller to work with, both in terms of software and hardware. Comparatively speaking, the ARM7 is smaller than the ARM9, costs less, is easier to solder, and requires fewer external components. Two solutions are presented for interfacing the TCM8230MD to a 32-bit ARM7 microcontroller.

15. SUBJECT TERMS

TCM8230MD, CMOS, camera, digital, video, microembedded

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Justin L. Shumaker
a. REPORT	b. ABSTRACT	c. THIS PAGE	UU	20	19b. TELEPHONE NUMBER (Include area code)
Unclassified	Unclassified	Unclassified			410-278-2834

Contents

List of Figures

1. Background

Just about any autonomous robotic system can benefit from a video capture device. However, digital video in the realm of meso- and microscale robotics is difficult to achieve. Often times, as the robotic platform size diminishes, so too do computational resources and sensor capabilities. This camera was selected as a result of the size, mass, and cost constraints of the robotic mobility platform. This size constraint greatly limited the selection of microcontrollers and cameras suitable for this application. This camera was originally designed for use in cell phones, hence, the small form factor and low cost. The microcontroller selected to interface this camera was the Atmel AT91SAM7S64. This microcontroller contains 16 kB of SRAM and 64 kB of Flash. It is capable of operating at a clock speed of 85 MHz. The objective of this project was to capture and process 128- $\times$ 96-pixel images from the camera at a frame rate greater than 10 frames per second (fps) to aid in stabilizing the attitude of the robotic platform during locomotion.

2. The Camera

The TCM8230MD complementary metal-oxide semiconductor (CMOS) camera module is a 20-pin integrated circuit (IC) that occupies a volume of $6 \times 6 \times 4.5$ mm (see figure 1). Using the camera's two-wire interface, more commonly known as inter-integrated circuit (I^2C), commands can be issued to the camera for setting various modes and parameters. The camera has a hexadecimal address of 0x60, and all communications to the camera consist of an 8-bit command followed by an 8-bit value. These commands manipulate the registers inside the camera's integrated microcontroller. In total, there are 95 of these commands to choose from controlling parameters such as image size, color, and gain. The data sheet does not provide details on what the vast majority of the registers do. There is only one command that must be sent to the camera for it to begin sending images—0x02. After issuing this command, the camera uses the *vd*, *hd*, *dout*, and *dclk* pins to transmit frame data. A transition from low to high on the *vd* pin indicates a new frame, while a transition from low to high on the *hd* pin indicates the beginning of a scanline. For every square waveform on the *dclk* pin, while both the *vd* and *hd* pins are high, there exist 8 bits of pixel data on the *dout* pins.[1]

---

[1]Toshiba CMOS Digital Integrated Circuit Silicon Monolithic TCM8230MD (A) Datasheet. Toshiba Ver 1.20; 1 April 2005, PDF.
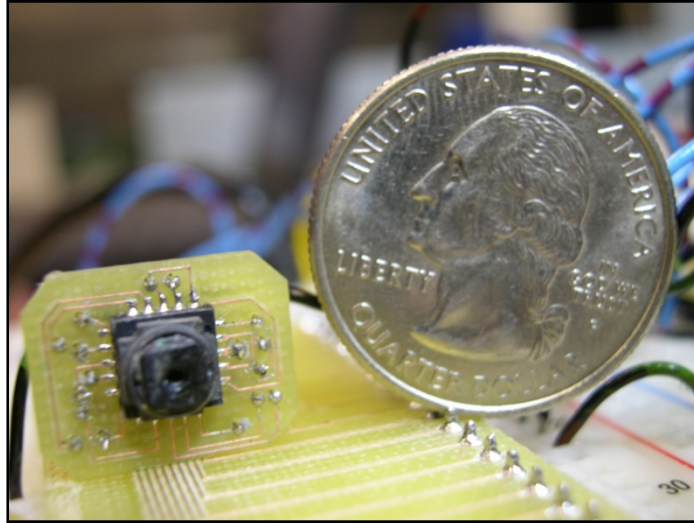
Figure 1. The TCM8230MD soldered to a printed circuit board.

The *vd* and *hd* pins are designed for driving interrupts on an external microcontroller. If the interrupts are configured for edge detection, then each transition on the *vd* and *hd* pins can be used to instruct the microcontroller to begin collecting data off the *dout* pins for each cycle on the *dclk* pin. This is accomplished through an interrupt service routine that executes upon the triggering of the interrupt. Interrupts used in this manner will permit the developer to utilize the time between frames to process the frame data. The code in appendix A uses three edge interrupts—FIQ, IRQ0, and IRQ1—to capture frame data. The FIQ pin is wired up to the *dclk* pin, while the IRQ0 and IRQ1 pins are wired up to the *vd* and *hd* pins, respectively. A more efficient implementation requiring only two interrupts is discussed later. The FIQ pin is the fast interrupt. There exists one of these pins on the ARM7[*] microcontroller. The purpose of the FIQ is to allow for an interrupt that utilizes as few clock cycles as possible. This is accomplished by having the interrupt code at a specific address on the microcontroller. The address that the microcontroller jumps to exists in hardware and cannot change. Therefore, it is possible to begin executing code from the FIQ handler within several clock cycles.[2]

Timing is very critical in this application. The faster the CPU can capture and process data, the faster the *extclk* pin on the camera can be driven to deliver frame data. The TCM8230MD data sheet indicates that a minimum of an 11.9-MHz square waveform must be sent to the *extclk* pin on the camera to drive its internal microcontroller. This is only partially true. After conducting several experiments, it was determined that the external clock need only run above ~6 MHz until the command 0x02 is issued. Afterwards, the *extclk* line can go well below 1 MHz before the camera stops functioning. Lowering the clock speed does several things. First and foremost, it voids any guarantee that the camera will function as described in the datasheet. Second, it reduces the frame rate of the camera. Third, it increases the exposure time, resulting in a more

---

[*]ARM is a registered trademark of ARM Ltd., Cambridge, England, UK.

[2]AT91SAM7S Series Preliminary Datasheet. 6175H-ATARM; Atmel Corporation, 3 December 2007.

washed out look. While the datasheet does not specify an equation that relates the external clock of the camera to a corresponding frame rate, it should be known that the frame rate is ~1.5 fps per 1 MHz. Applications requiring 30 fps will require an external clock of ~20 MHz. Fourth, it increases the amount of time available for executing the instructions necessary to capture the frame data. Fifth, it leads to blur in the presence of motion. The *dclk* pin outputs at a frequency of *extclk*/2 (figure 2). For example, a 4-MHz square waveform is sent to the external clock pin on the camera. Then, the *dclk* will be running at 2 MHz. If data is transmitting at a rate of 2 MHz, this only allows for a 500-ns window of time to execute the necessary instructions to buffer the pixel data into external memory for storage. At 4 MHz, this will provide ~6 fps.
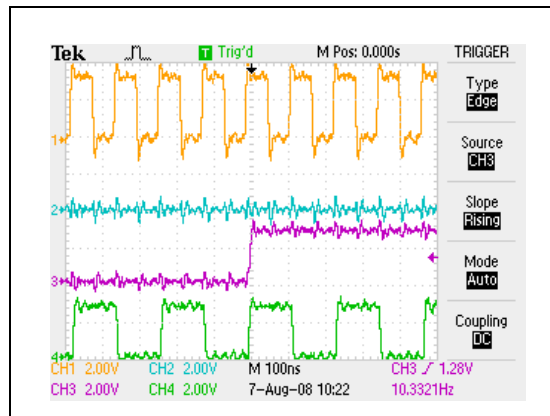

Figure 2. Yellow and green display, *extclk* and *dclk*, respectively.

The camera allocates a fixed window of time for each horizontal scanline independent of the resolution. Larger scanlines require more of the window to deliver pixel data, while smaller scanlines require less. The remaining time in each window is very advantageous. This time is used to format the pixel data. The implementation in appendix A demonstrates how a 128- $\times$ 96-pixel image is captured, formatted, and stored. The pixel data is received in a 16-bit RGB:565 format. Each pair of bytes consists of five red bits, six green bits, and five blue bits. This color format is the result of the bayer configuration of the cameras CMOS matrix. Because RGB:565 pixel data requires 2 bytes per pixel, a 256-byte buffer exists for each 128-pixel scanline. The remaining time in each scanline window is used to convert the RGB:565 pixel data into 7-bit grayscale image data using the seven most significant bits in a byte. The grayscale image is 7 bits because of the sum of the depths of each color channel.

32 red + 128 green + 32 blue = 128 total   ..1
$Log_2 128 = 7$ bits   ..2

One should note that this image format is most influenced by the green channel. Other grayscale formats can be employed, but this method provided the best results. Once the scanline data is formatted, whether it be color or grayscale, it is stored in an array in memory and a scanline index is incremented. The scanline index is set to 0 when the camera's *vd* pin goes high to indicate the beginning of a new frame and the process repeats.

It is also worth pointing out that 7-bit grayscale images always have 0 as the last bit in each byte. This bit can be used for image compression. If two adjacent bytes are identical, then the first byte can set the least significant bit to 1 and the second byte discarded. On average, this form of lossless compression displayed a 25%–30% image size reduction.

3. A More Efficient Buffering Algorithm

By leveraging the fact that the camera transmits data as a function of its external clock frequency, one can create a set of timed instructions that remove the overhead of interrupt-driven programming. This not only creates a more efficient algorithm but provides the facility to process nearly twice the frame rate, 10.3 fps. Recall that the camera frequency is a function of the microcontroller's frequency. Therefore, it is known how many microcontroller clock cycles will occur between pixel data bytes from the camera. Using this information, one can develop an assembly algorithm that has a specific number of instructions, corresponding to a specific number of clock cycles, which are timed such that each iteration of these instructions guarantees the capture of sequential pixel data bytes. To elaborate on this strategy, the details of the algorithm will be discussed. The ARM7 master clock frequency was configured to operate at 84,787,200 Hz. A pulse width modulation square wave was generated by the ARM7 to the cameras *extclk* pin at 8,478,720 Hz. Therefore, the *dclk* frequency from the camera is 4,239,360 Hz. It can be seen that 20 clock cycles on the microcontroller must be used for every pixel data byte from the camera. Less than 10 instructions can be processed during this very short time (235.88 ns). To complicate matters, not all instructions consume the same number of clock cycles. Even worse, there is a three-stage pipeline in which instructions are processed. A deep understanding of the ARM7 architecture and assembly instructions is required to make this strategy work. Several instructions must occur between each pixel byte. They include reading the pin data status register, shifting the data bits such that they occupy the lowest 8 bits of the 32-bit register, and storing the register byte into memory. Unlike the previous algorithm, the scanline capture code and pixel component averaging exists exclusively in assembly. These instructions, along with the rest of the instructions, can be seen in appendix B. An example image captured at a resolution of 128×96 pixels in 7-bit grayscale can be seen in figure 3. A somewhat modified version of this algorithm can permit an additional doubling of the frame rate by instructing the camera to output twice the resolution and throwing away every other pixel. The details of this method will not be discussed in this report.

4

Figure 3. 128- × 96-pixel 7-bit
grayscale image.

4. Conclusion

It is indeed possible to capture color or grayscale image data at a reasonable frame rate from an I^{2}C 8-bit CMOS camera using an ARM7 microcontroller. The software necessary to interface the camera may be more complicated than that of a microcontroller with dedicated hardware for I^{2}C 8-bit cameras. However, the circuit board will have lower cost and complexity. If size or cost do not matter, then it is certainly recommended that an ARM9 or other microcontroller with dedicated camera hardware be used instead. For higher resolutions, Toshiba manufactures the TCM8240MD, a 1.3-MP camera with onboard JPEG compression.

INTENTIONALLY LEFT BLANK.

Appendix A.   Assembly Routine With DCLK Interrupt

This appendix appears in its original form, without editorial change.

crt.s (excerpt)

```
/**************************************************************************
* Function: AT91F_Fiq_Handler
*
* Programmer: Justin Shumaker
**************************************************************************/
AT91F_Fiq_Handler:

/* Ideas: Pull r12 and r10 out of the interrupt since they never change, but they must be initialized in FIQ mode */

/* Use address R12 as the basis for addressing SODR, CODR, and ODSR */
/* Read the AIC Fast Interrupt Vector register to clear the interrupt */
        ldr     r12, =AT91C_BASE_AIC
        ldr     r11, [r12, #AIC_FVR]

/* Ver2 Buffer Data - Load contents of PDSR into R11 */
        ldr     r9, [r12, #0x43C]

/* IDEA: Consider removing this block and just disabling the interrupt while PA30 is low */
/* Return if the HD line (PA30 = #0x40000000) is low */
        tst     r9, #0x40000000
        subeqs  pc, lr, #4


/* Data from camera is in bits 20..28 of PDSR register, it needs to be in bits 0..7, logical shift right 21 bits. */
        mov     r11, r9, lsr #21

/* Buffer Data - Store data into hbuf */
        ldr     r10, =hbuf

/* Increment r7 by 1 and stop at 256 (byte 257) */
        strb    r11, [r10, r7]
        cmp     r7, #256
        addlt   r7, r7, #1

/* Toggle PA8 for debugging */
/*
        mov     r8, #0x100
        str     r8, [r12, #0x434]
        str     r8, [r12, #0x438]
*/

/* Return from Fiq interrupt */
        subs    pc, lr, #4
```

main.c

```c
#include "board.h"
#include "pio.h"
#include "lowlevel.h"
#include "usart.h"
#include "pwm.h"
#include "aic.h"
#include "twi.h"
#include "usart.h"
#include <stdio.h>

unsigned char hbuf[257]; /* 2B per pixel (128) + 1 byte overflow */
unsigned char image[12288]; /* 128x96 8-bit grayscale image */
int h_offset;


/* Negative Edge = Frame is done */
static void
int_vd_isr (void)
{
  if (h_offset >= 12288)
```

```c
  {
    AT91C_BASE_AIC->AIC_IDCR = (1<<AT91C_ID_IRQ0);
    AT91C_BASE_AIC->AIC_IDCR = (1<<AT91C_ID_IRQ1);
    AT91C_BASE_AIC->AIC_IDCR = (1<<AT91C_ID_FIQ);
    h_offset = 99999; /* Ensure the interrupt has acknowledged the image is completed and disable in case main ()
gets to it first. */
  }

  /* Idea: disable FIQ while VD is low... */
#if 1
  if  ((AT91C_BASE_PIOA->PIO_ODSR & LED_STATUS) == LED_STATUS)
  {
    AT91C_BASE_PIOA->PIO_CODR = LED_STATUS; /* turn status on */
  }
  else
  {
    AT91C_BASE_PIOA->PIO_SODR = LED_STATUS; /* turn status led off */
  }
#endif
}


static void
int_hd_isr (void)
{
  int i;

  /*
  * On the falling edge of HD the scanline has been completely buffered.
  * March through the scanline and stuff into image buffer.
  * Reset any flags and indices for next scanline.
  */

  /* Reset hbuf index to 0 */
  asm volatile ("mov r7, #0" : :);


  for (i = 0; i < 128; i++)
  {
    /* RGB565 to grayscale */
    image[h_offset+i] = ((hbuf[2*i] & 0x1F) + ((hbuf[2*i]>>5) | (hbuf[2*i+1] & 0x7)<<3) + (hbuf[2*i+1] >> 3)) << 1;

    /* Green only */
/*    image[h_offset+i] = ((hbuf[2*i]>>5) | (hbuf[2*i+1] & 0x7)<<3) << 2; */

    /* Red only */
//    image[h_offset+i] = (hbuf[2*i] & 0x1F)<<3;
  }

   /* Is this really necessary? */
  if (h_offset < 12288)
    h_offset += 128;

}


int
main (void)
{
  int i, n;
  unsigned char data[2];

  low_level_init (EXT_OSC, PLL_DIV, PLL_MUL, PRESCALE);

  usart_init (USART0, 57600, AT91C_US_CHRL_8_BITS, AT91C_US_PAR_NONE, AT91C_US_NBSTOP_1_BIT, MCK);

  /* Peripheral A Enable = PWM2, Peripheral B Enable = FIQ */
  pio_init (LED_STATUS|PA2|PA21|PA22|PA23|PA24|PA25|PA26|PA27|PA28, PA0|PA5|PA6|PA30, PA19|PA20, LED_STATUS|PA2,
LED_STATUS, 0, 0, 0, 0);

  /* Start the ext clock at 14.335 MHz */
  pwm_init (PWM_CH0, 0, 0, 0, 0);
```

```c
  PWM_CH0_MODE = PWM_MCK;
  PWM_CH0_PERIOD = 6;
  PWM_CH0_DUTY = 3;

  /* Delay */
  for (i = 0; i < 5000; i++);
/* usart_send (USART0, 17, "Program Started\r\n"); */

  /* Enable the Camera */
  twi_init (39, 39, 0x1);

  /* Color Bar Test */
//  data[0] = 0x1E;
//  data[1] = 0x6D;
//  twi_send (0x3C, data, 2, 1);

  data[0] = 0x03; /* Register Address */
  data[1] = 0x26; /* Enable Output @ 128x96 in RGB565 */
  twi_send (0x3C, data, 2, 1);

  /* Drop the ext clock to 3.584 MHz (24,12) */
  PWM_CH0_PERIOD = 24;
  PWM_CH0_DUTY = 12;

  /* Initialize to 0 */
  h_offset = 0;
  asm volatile ("mov r7, #0" : :);


  /* Feed output of PWM into interrupt */
  aic_int_enable (0, AT91C_ID_FIQ, 0, AT91C_AIC_SRCTYPE_EXT_NEGATIVE_EDGE, 0);
  aic_int_enable (int_hd_isr, AT91C_ID_IRQ1, 0, AT91C_AIC_SRCTYPE_EXT_NEGATIVE_EDGE, 0);
  aic_int_enable (int_vd_isr, AT91C_ID_IRQ0, 0, AT91C_AIC_SRCTYPE_EXT_NEGATIVE_EDGE, 0);

  n = 0;
  while (1)
  {
    if (h_offset == 99999)
    {
      h_offset = 0;

      /* TRANSMIT DATA TO COMPUTER */

      if (!(n % 15))
        usart_send (USART0, 12288, image);

      n++;

      /* If doing something very time consuming then must wait until vd goes low again */
//      while (AT91C_BASE_PIOA->PIO_PDSR & PA20);

      AT91C_BASE_AIC->AIC_IECR = (1<<AT91C_ID_FIQ);
      AT91C_BASE_AIC->AIC_IECR = (1<<AT91C_ID_IRQ0);
      AT91C_BASE_AIC->AIC_IECR = (1<<AT91C_ID_IRQ1);
    }
  }
}
```

Appendix B. Assembly Routine Without DCLK Interrupt

This appendix appears in its original form, without editorial change.

crt.s (excerpt)

```
/***************************************************************************
* Function: AT91F_Fiq_Handler
*
* Programmer: Justin Shumaker
***************************************************************************/
AT91F_Fiq_Handler:

/* Ideas: Pull r12 and r10 out of the interrupt since they never change, but they must be initialized in FIQ mode */

/* Use address R12 as the basis for addressing SODR, CODR, and ODSR */
/* Read the AIC Fast Interrupt Vector register to clear the interrupt */
        ldr     r12, =AT91C_BASE_AIC

/* Load hbuf address into r10 */
        ldr     r10, =hbuf
        mov     r7, #0

        mov     r7, #0
        mov     r7, #0
        mov     r7, #0
        mov     r7, #0
        mov     r7, #0
        mov     r7, #0
        mov     r7, #0
        mov     r7, #0
        mov     r7, #0


foobar1:
/* Buffer Data - Load contents of PDSR into R11 */
        ldr     r9, [r12, #0x43C]

/* Data from camera is in bits 20..28 of PDSR register, it needs to be in bits 0..7, logical shift right 21 bits. */
        mov     r11, r9, lsr #21

        strb    r11, [r10, r7]          /* Store byte into hbuf with offset */

        add     r7, r7, #1              /* Increment Counter */
        cmp     r7, #255            /* Stop after 256 iterations */

        bne     foobar1                 /* Loop */


/* BLINK */
/*
mov     r8, #0x100
str     r8, [r12, #0x434]
str     r8, [r12, #0x438]
*/

ldr     r12, =AT91C_BASE_AIC
ldr     r11, [r12, #AIC_FVR]

/* Load h_offset address into r11 */
        ldr     r11, =h_offset
        ldr     r9, [r11]

/* Ensure that we don't overflow the buffer because h_offset was not reset correctly */
        cmp     r9, #12288
        subges  pc, lr, #4

/* Reset r7 counter to 0 and load address of image into r8 */
        mov     r7, #0
        ldr     r8, =image

/*
```

```
* CURRENT REGISTER USAGE:
* R12 = AIC BASE
* R11 = h_offset address
* R10 = hbuf
* R09 = h_offset value
* R08 = image
* Using R11, R12, R13
*/


foobar2:
/* r10 contains hbuf byte to be stored in image */
        ldrb    r11, [r10, r7]          /* Load every nth byte of hbuf (pixel row data) into r0 */

/* image[h_offset+i] = ((hbuf[2*i] & 0x1F) + ((hbuf[2*i]>>5) | (hbuf[2*i+1] & 0x7)<<3) + (hbuf[2*i+1] >> 3)) << 1; */
        and     r12, r11, #0x1F         /* 5-bits Red in R1 */

        mov     r11, r11, asr #5  /* 3-bits Green LSB shift right 5 */
        add     r12, r12, r11

        add     r7, r7, #1              /* Increment */
        ldrb    r11, [r10, r7]          /* Load every n+1th byte of hbuf (pixel row data) into r1 */

        and     r13, r11, #7            /* 3-bits Green MSB shift left 3 */
        mov     r13, r13, asl #3
        add     r12, r12, r13

        mov     r11, r11, asr #3  /* 5-bits Blue from R1 shift right 3 */
        add     r12, r12, r11

        mov     r12, r12, asl #1  /* left shift 1-bit so it's MSB oriented */

        add     r13, r9, r7, asr #1     /* Compute offset h_offset + (r7)/2 */
        strb    r12, [r8, r13]          /* Store r12 at image address + offset r1 */

        add     r7, r7, #1              /* Increment Counter */
        cmp     r7, #256        /* Stop after 128 iterations */

        bne     foobar2                 /* Loop */


/* Increment h_offset by 128 */
        add     r9, r9, #128
        ldr     r11, =h_offset
        str     r9, [r11]



/* Used for testing data capture timing alignment */
/*
mov     r8, #0x100
str     r8, [r12, #0x434]
str     r8, [r12, #0x438]
*/
subs    pc, lr, #4
```

NO. OF
COPIES ORGANIZATION

  1   DEFENSE TECHNICAL
 (PDF   INFORMATION CTR
 only)   DTIC OCA
          8725 JOHN J KINGMAN RD
          STE 0944
          FORT BELVOIR VA 22060-6218

  1   DIRECTOR
          US ARMY RESEARCH LAB
          IMNE ALC HR
          2800 POWDER MILL RD
          ADELPHI MD 20783-1197

  1   DIRECTOR
          US ARMY RESEARCH LAB
          AMSRD ARL CI OK TL
          2800 POWDER MILL RD
          ADELPHI MD 20783-1197

  1   DIRECTOR
          US ARMY RESEARCH LAB
          AMSRD ARL CI OK PE
          2800 POWDER MILL RD
          ADELPHI MD 20783-1197

          ABERDEEN PROVING GROUND

  1   DIR USARL
          AMSRD ARL CI OK TP (BLDG 4600)

14