Hi all

I have a STM32F429i Discovery board (version 0), and a MT9M001 camera. I am trying to save the picture to SDRAM, to do further processing on later. The camera is 10b. However, I believe I am having two main issues.

My setup:
MT9M001 -> ebay breakout board -> STM32F429i Disc0 via bundled jumper leads. Connected to the DCMI pins, and alt-functions set.
DCMI Config:
Using appropriate polarity's & such. am successfully generated line event interrupts, and frame complete interrupts. Also checking for overrun interrupts - these only seem to occur in the time between the camera initializing and DMA being setup.

SDRAM initialized, and put into linker script like:

```
MEMORY
{
 ram (rwx) : ORIGIN = 0x20000000, LENGTH = 192K
 rom (rx) : ORIGIN = 0x08000000, LENGTH = 2048K
 ccm (rwx) : ORIGIN = 0x10000000, LENGTH = 64K
 sdram (rx) : ORIGIN = 0xD0000000, LENGTH = 8192K
}
/* skip a few lines - after the ccm definition */
 /* Added some! to get the SDRAM working */
 .sdram : {
    *(.sdramtext)
    *(.sdramtext*)
    *(.sdramrodata)
    *(sdramrodata*)
} >sdram
```

This allows me to use *__attribute__((section(".sdram")))* to put stuff onto SDRAM, which appears to be working.

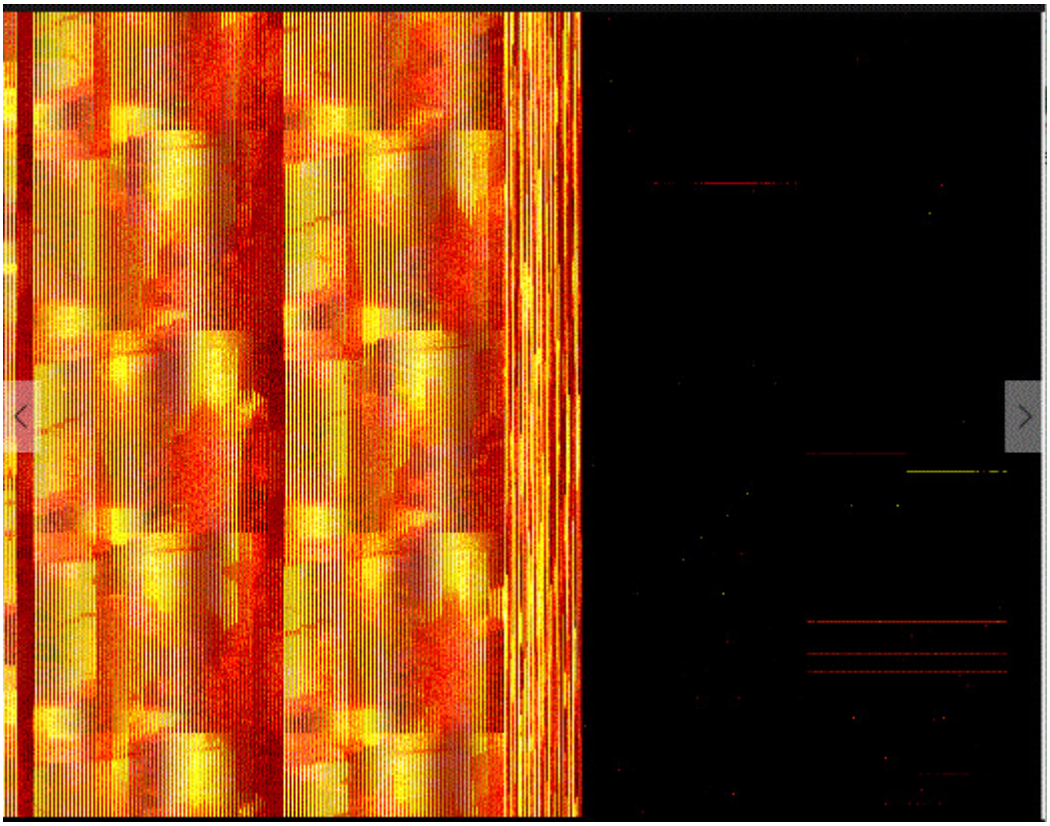
DMA config:
This is the part I have the most trouble with.
Using peripheral to memory, with a transfer data size of 32b (as the DCMI DR is 32b, and the DCMI peripheral packs the two 16b pixels into it). Using FIFO (with full threshold), but only single (burst) mode for both peripheral and memory.

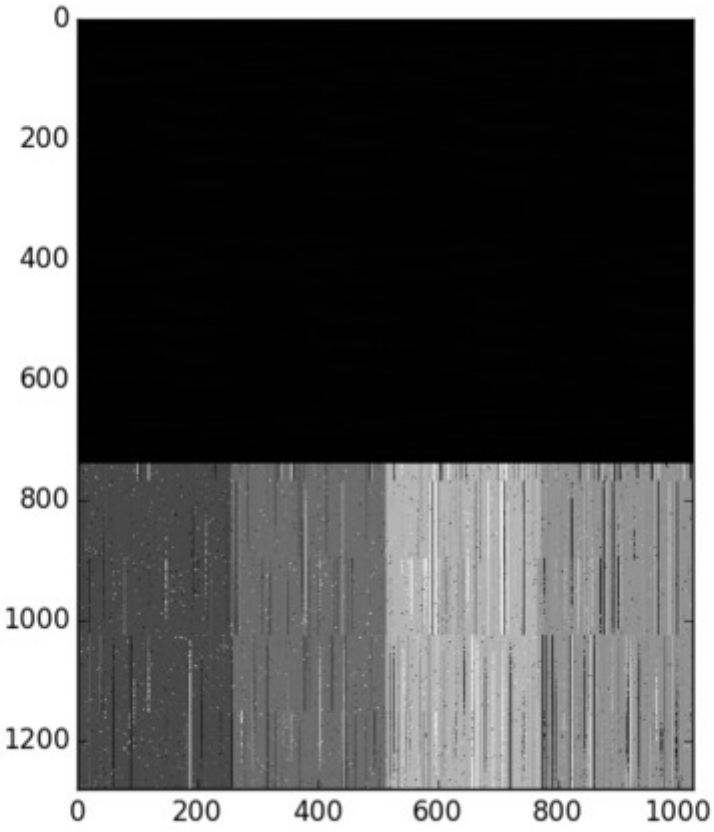
The memory address is somewhat complicated. I am using double buffering mode - but as the camera is 1 MP (1.3 million pixels ~ 625k DMA transfers) I cannot use just the two buffers, as the DMA can only transfer up to 2^16 data packet/things per buffer. So I have a DMA transfer complete (DMA TC) interrupt, which I use to march the address through the SDRAM. This, to some extent, seems to be working - I haven't checked it all, but it does increment correctly. As seen in the pictures, I am getting bad data at points though.

Camera clock source:
As yet, I have been rather lazy, and not clocked the camera past 2.6MHz. The datasheet does say it is suitable for 1->48MHz, so that shouldn't be a problem?

Issue 1: Synchronisation
I have made a rather poor attempt at synchronising the DMA and camera. As seen in the picture, it just doesn't work.

Issue 2: Image data approximately invariant
Whilst the particular image data does change, it seems that the bar pattern is consistent. Additionally, the data does not seem to change when I shine a torch into the camera.
Two example pictures





The code:
Header:

```
#ifndef MT9M001
#define MT9M001
#include <stdint.h> //provides types such as uintX_t
#include <string.h>

#include "i2cMaster.h"
#include "../configuration.h"
```

```
08. #include "sdram.h"
09.
10. #include <libopencm3/stm32/rcc.h>
11. #include <libopencm3/stm32/gpio.h>
12. #include <libopencm3/stm32/timer.h>
13. #include <libopencm3/stm32/dma.h>
14. #include <libopencm3/cm3/nvic.h>
15.
16.
17. #define SLAVE_ADDRESS 0x5d //0b1011101 - as the write address is given as 0xBA
18.
19. #define MAXROWLENGTH 1024
20. #define MAXHEIGHT 1280
21.
22. extern uint16_t rowLength;
23.
24. static uint16_t row = 0;
25.
26. extern uint32_t __attribute__((section(".sdram"))) rawPicture[MAXHEIGHT][MAXROWLENGTH/2];
27. extern uint16_t __attribute__((section(".sdram"))) pictureBuf[MAXHEIGHT][MAXROWLENGTH];
28.
29. static uint8_t oldBank = 0;
30.
31. #define BUFLENGTH 0xA000//0x400//a row//0xA000//32 rows
32. static uint16_t chunk = 0;
33. static uint16_t rowBuf1[BUFLENGTH];
34. static uint16_t rowBuf2[BUFLENGTH];
35.
36. //Do we want to actually save the picture?
37. //Really really basic synchronisation method
38. static uint8_t savePicture = 0;
39.
40. #define STATUSPIN GPIO7
41. #define STATUSPIN2 GPIO9
42.
43.
44. //DCMI is not included in libopencm3 => hence do it by hand
45. typedef struct {
46.     uint32_t CR;
47.     uint32_t SR;
48.     uint32_t RIS;
49.     uint32_t IER;
50.     uint32_t MIS;
51.     uint32_t ICR;
52.     uint32_t ESCR;
53.     uint32_t ESUR;
54.     uint32_t CWSTRT;
55.     uint32_t CWSIZE;
56.     uint32_t DR;
57. } DCMI_HW_Registers;
58.
59.
60. void camSetup(void);
61.
62. void camConfig(void);
63.
64.
65. void clkinSetup(void);
66.
67. void camDCMISetup(void);
68.
69.
```

```
70. #endif
```

Source:

```
001. #include "mt9m001.h"
002.
003. uint32_t __attribute__((section(".sdram"))) rawPicture[MAXHEIGHT][MAXROWLENGTH/2];
004. uint16_t __attribute__((section(".sdram"))) pictureBuf[MAXHEIGHT][MAXROWLENGTH];
005.
006. uint16_t rowLength = MAXROWLENGTH;
007.
008.
009. void camSetup(void){
010.
011.
012.     //DCMI (the camera interface) is spread out rather messily over several different banks
of pins
013.     //using alt-function mode 13
014.
015.     //Data inputs D0 -> D9
016.
017.     //D0, D1 are on bank A
018.     rcc_periph_clock_enable(RCC_GPIOA);
019.     gpio_mode_setup(GPIOA, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO9 | GPIO10);
020.     gpio_set_af(GPIOA, GPIO_AF13, GPIO9 | GPIO10);
021.     //D2, D3 are on bank B
022.     rcc_periph_clock_enable(RCC_GPIOB);
023.     gpio_mode_setup(GPIOB, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO8 | GPIO9);
024.     gpio_set_af(GPIOB, GPIO_AF13, GPIO8 | GPIO9);
025.     //D4, D8, D9 are on bank C
026.     rcc_periph_clock_enable(RCC_GPIOC);
027.     gpio_mode_setup(GPIOC, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO10 | GPIO11 | GPIO12);
028.     gpio_set_af(GPIOC, GPIO_AF13, GPIO10 | GPIO11 | GPIO12);
029.     //D5 is on bank D
030.     rcc_periph_clock_enable(RCC_GPIOD);
031.     gpio_mode_setup(GPIOD, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO3);
032.     gpio_set_af(GPIOD, GPIO_AF13, GPIO3);
033.     //D6, D7 are on bank E
034.     rcc_periph_clock_enable(RCC_GPIOE);
035.     gpio_mode_setup(GPIOE, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO5 | GPIO6);
036.     gpio_set_af(GPIOE, GPIO_AF13, GPIO5 | GPIO6);
037.
038.     //Setup camera outputs used by DCMI
039.     //      Vsync, Hsync, PIX_CLK
040.
041.     //HS and PIX_CLK are on bank A
042.     rcc_periph_clock_enable(RCC_GPIOA);
043.     gpio_mode_setup(GPIOA, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO4 | GPIO6);
044.     gpio_set_af(GPIOA, GPIO_AF13, GPIO4 | GPIO6);
045.     //VS is on bank B
046.     rcc_periph_clock_enable(RCC_GPIOB);
047.     gpio_mode_setup(GPIOB, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO7);
048.     gpio_set_af(GPIOB, GPIO_AF13, GPIO7);
049.
050.
051.
052.     //Set the outputs (inputs to camera)
053.     //The camera inputs are:
054.     //      XVCLK:  XCK:    CLKIN:      PB0:    The master clock for the camera. See
clkinSetup()
055.     //      /Reset: RST:    Reset_bar:  PC1:        On LOW resets sensor (resets registers)
056.     //      Standby:SBY:                PC2:        On HIGH disables analog circuitry for
power saving

057.     //      Trigger:TRG:                PC3:        On High, activates snapshot sequence.
Hold low during CTS mode
058.     rcc_periph_clock_enable(RCC_GPIOC);
```

```
059.     gpio_mode_setup(GPIOC, GPIO_MODE_OUTPUT, GPIO_PUPD_PULLDOWN, GPIO1| GPIO2 | GPIO3);
060.     gpio_set_output_options(GPIOC, GPIO_OTYPE_PP, GPIO_OSPEED_50MHZ, GPIO1| GPIO2 | GPIO3);
061.     //Set initial state:
062.     //      XVCLK       Handled by timer
063.     //      /Reset      LOW, wait, then HIGH
064.     //      Standby     LOW
065.     //      Trigger     LOW
066.     gpio_clear(GPIOC, GPIO1); // reset sensor
067.     gpio_clear(GPIOC, GPIO2); //Set analog circuitry on
068.     gpio_clear(GPIOC, GPIO3); //Set to CTS mode
069.     for (int i=0; i<2000000; i++){
070.         __asm__("nop");
071.     }
072.     gpio_set(GPIOC, GPIO1); //finish reset
073.
074.
075.     //Setup I2C control interface
076.     //Using I2C3
077.     //  SDA:    PC9
078.     //  SCL:    PA8
079.     //SCL first
080.     rcc_periph_clock_enable(RCC_GPIOA);
081.     gpio_mode_setup(GPIOA,
082.                     GPIO_MODE_AF,
083.                     GPIO_PUPD_NONE, //Mt9M001 development board has built in pullups
084.                     GPIO8);
085.     gpio_set_output_options(GPIOA, GPIO_OTYPE_OD, GPIO_OSPEED_50MHZ, GPIO8); //Must set
this - otherwise will not allow acks
086.     gpio_set_af(GPIOA, GPIO_AF4, GPIO8);
087.     //SDA
088.     rcc_periph_clock_enable(RCC_GPIOC);
089.     gpio_mode_setup(GPIOC,
090.                     GPIO_MODE_AF,
091.                     GPIO_PUPD_NONE, //MT9M001 development board has built in pullups
092.                     GPIO9);
093.     gpio_set_output_options(GPIOC, GPIO_OTYPE_OD, GPIO_OSPEED_50MHZ, GPIO9); //Must set
this - otherwise will not allow acks
094.     gpio_set_af(GPIOC, GPIO_AF4, GPIO9);
095.
096.     rcc_periph_clock_enable(RCC_I2C3);
097.
098.
099.
100.     //Setup a pin for debugging
101.     rcc_periph_clock_enable(RCC_GPIOF);
102.     gpio_mode_setup(GPIOF, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, STATUSPIN | STATUSPIN2);
103.     gpio_set_output_options(GPIOF, GPIO_OTYPE_PP, GPIO_OSPEED_100MHZ, STATUSPIN |
STATUSPIN2);
104.     gpio_set(GPIOF, STATUSPIN | STATUSPIN2);
105.
106.
107.
108.
109.
110.     SDRAMSetup();
111.
112.     //Clear the picture
113.     for (int i=0;i<MAXROWLENGTH/2;i++){
114.         for (int j=0;j<MAXHEIGHT/2;j++){
115.             rawPicture[i][j] = i;
116.         }
117.     }
118.
119.     clkinSetup(); //Clock must be setup before i2c communication is attempted
120.
121.     initI2C3();
```

```
122.     camConfig();
123.
124.
125.
126.     camDCMISetup();
127. }
128.
129. void camConfig(void){
130.     //Use the SCCB interface to the camera to configure it
131.     //Use MT9M001_registers.pdf - note that the
132.     //register map has been removed from newer versions
133.     //of the datasheet
134.
135.     //Reset sensor via i2c
136.     I2CStart(I2C3, SLAVE_ADDRESS, I2C_WRITE);
137.     //write a 1 to LSB of 0x0D register
138.     I2CWrite(I2C3, 0x0D); //Write to reset register 0x0D
139.     I2CWrite(I2C3, 0x0);
140.     I2CWrite(I2C3, 0x1);
141.     //Write a 0 to LSB of 0x0D register
142.     I2CStart(I2C3, SLAVE_ADDRESS, I2C_WRITE);
143.     I2CWrite(I2C3, 0x0D); //Write to reset register 0x0D
144.     I2CWrite(I2C3, 0x0);
145.     I2CWrite(I2C3, 0x0);
146.
147.     //Read the chip version
148.     //  Chip version should be 0x8431. This corresponds to the monocrhome version of the
sensor - whereas 0x8411 is the colour
149.
150.     I2CStart(I2C3, SLAVE_ADDRESS, I2C_WRITE);
151.     I2CWrite(I2C3, 0x00); //Chip version register
152.     I2CStart(I2C3, SLAVE_ADDRESS, I2C_READ);
153.     uint8_t d1 = I2CReadAck(I2C3);
154.     uint8_t d2 = I2CReadNack(I2C3); //read nack as we want to close
155.     uint16_t chipVersion = (d1 <<8) + d2;
156.
157.
158.
159.     //Chip Enable
160.     I2CStart(I2C3, SLAVE_ADDRESS, I2C_WRITE);
161.     I2CWrite(I2C3, 0xF1); //chip enable register
162.     //write a 1 to LSB of chip enable register
163.     I2CWrite(I2C3, 0x0);
164.     I2CWrite(I2C3, 0x1);
165.     I2CStop(I2C3);
166.
167.     //Double check that we put the correct value in
168.     I2CStart(I2C3, SLAVE_ADDRESS, I2C_WRITE);
169.     I2CWrite(I2C3, 0xF1); //chip enable register
170.     I2CStart(I2C3, SLAVE_ADDRESS, I2C_READ);
171.     uint8_t d3 = I2CReadAck(I2C3);
172.     uint8_t d4 = I2CReadNack(I2C3);
173.     I2CStop(I2C3);
174.
175.     //Check the value of the "Output Control" 0x07 register
176.     //  Bit 0: 0=normal operation
177.     //  Bit 1: 1 =normal operation
178.     //  Bit 2: Reserved
179.     //  Bit 3: Reserved
180.     //  Bit 6: Override pixel data. 0 = normal operation
181.     I2CStart(I2C3, SLAVE_ADDRESS, I2C_WRITE);
182.     I2CWrite(I2C3, 0x07);
183.     I2CStart(I2C3, SLAVE_ADDRESS, I2C_READ);

184.     uint8_t d5 = I2CReadAck(I2C3);
185.     uint8_t d6 = I2CReadNack(I2C3);
186.     I2CStop(I2C3);
```

```
187.
188.
189.     //Set the size of the captured image
190.     //Wndow Control - registers 0x01 -> 0x04
191.     I2CStart(I2C3, SLAVE_ADDRESS, I2C_WRITE);
192.     I2CWrite(I2C3, 0x01); //First row to be read out
193.     I2CWrite(I2C3, 0x0);
194.     I2CWrite(I2C3, 0xC);
195.     //Mt9m001 auto-increments register address
196.     //hence can write into 0x02
197.     I2CWrite(I2C3, 0x0);
198.     I2CWrite(I2C3, 0x14);
199.     //hence can write into 0x03
200.     I2CWrite(I2C3, 0x03);
201.     I2CWrite(I2C3, 0xFF);
202.     //hence can write into 0x04
203.     I2CWrite(I2C3, 0x4);
204.     I2CWrite(I2C3, 0xFF);
205.     I2CStop(I2C3);
206.
207.
208. }
209.
210. void clkinSetup(void){
211.
212.     //PB0 is clock signal for camera
213.
214.     //Configure gpio
215.     rcc_periph_clock_enable(RCC_GPIOB);
216.     gpio_mode_setup(GPIOB, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO0);
217.     gpio_set_output_options(GPIOB, GPIO_OTYPE_PP, GPIO_OSPEED_100MHZ, GPIO0);
218.     //Set AF so the timer is straight to the pin
219.     gpio_set_af(GPIOB, GPIO_AF2, GPIO0); //TIM3_CH3 is AF2 for PB0
220.
221.     //Configure timer
222.     rcc_periph_clock_enable(RCC_TIM3);
223.
224.     timer_reset(TIM3);
225.
226.     timer_set_mode(TIM2,
227.                    TIM_CR1_CKD_CK_INT, //internal base clock
228.                    TIM_CR1_CMS_EDGE, //edge count
229.                    TIM_CR1_DIR_UP); //count up
230.     timer_set_prescaler(TIM2, 0); //no prescaler = 0
231.
232.     timer_set_period(TIM3, 2);
233.
234.
235.     // Disable outputs.
236.     timer_disable_oc_output(TIM3, TIM_OC1);
237.     timer_disable_oc_output(TIM3, TIM_OC2);
238.     timer_disable_oc_output(TIM3, TIM_OC3);
239.     timer_disable_oc_output(TIM3, TIM_OC4);
240.
241.     //Configure channel 3 for PB0 (TIM3)
242.     timer_disable_oc_clear(TIM3, TIM_OC3);
243.     timer_disable_oc_preload(TIM3, TIM_OC3);
244.     timer_set_oc_slow_mode(TIM3, TIM_OC3);
245.     timer_set_oc_mode(TIM3, TIM_OC3, TIM_OCM_TOGGLE);//TIM_OCM_TOGGLE);
246.     //unknown which may make it work
247.     timer_enable_oc_output(TIM3, TIM_OC3);
248.     timer_set_oc_polarity_high(TIM3, TIM_OC3);
249.
250.     /* Set the capture compare value for OC3. */
251.     timer_set_oc_value(TIM3, TIM_OC3, 1);
252.
```

```
253.     //ARR reload enable
254.     timer_disable_preload(TIM3);
255.
256.     timer_enable_counter(TIM3);
257. }
258.
259.
260. void camDCMISetup(void){
261.     //Setup the DCMI camera controller to download pictures
262.     //DCMI is not supported in libopencm3 => do it with registers
263.
264.     //Enable DCMI clock
265.     rcc_periph_clock_enable(RCC_DCMI);
266.
267.     volatile DCMI_HW_Registers *dcmi = (DCMI_HW_Registers *) DCMI_BASE; //DCMI_BASE
provided by libopencm3
268.
269.     //Disable DCMI
270.     dcmi->CR &= ~(1 << 14);
271.
272.
273.     //Configure DCMI
274.     //Reset register
275.     dcmi->CR &= 0x0000;
276.     //Set to 10-bit data by settings bits 10, 11 to 01
277.     dcmi->CR |= (0b01<<10);
278.     //Frame capture rate control - what proportion of frames to capture
279.     //      10: 25% of frames captured
280.     //      00: all frames captured
281.     //dcmi->CR |= (0b10<<8);
282.     //Vsync - what level on vsync pin for not valid?
283.     //      For MT9M001, valid when high => set this as low
284.     dcmi->CR &= ~(1<<7);
285.     //Hsync. What level on hsync pin for not valid?
286.     //      For MT9M001, when low, not valid
287.     dcmi->CR &= ~(1<<6);
288.     //PIX_CLK polarity. Configures capture edge of pixel clock
289.     //      For MT9M0001, capture data on falling edge
290.     dcmi->CR &= ~(1<<5);
291.     //Hardware or embedded synchronisation?
292.     //      For MT9M001, hardware synchronisation
293.     dcmi->CR &= ~(1<<4);
294.     //JPEG FORMAT?
295.     //      For MT9M0001, No.
296.     dcmi->CR &= ~(1<<3);
297.     //Crop feature. Crop the image or not
298.     dcmi->CR &= ~(1<<2); //No
299.     //Continous capture mode? Recieved data is put into a buffer continously
300.     dcmi->CR &= ~(1<<1); //Continous
301.
302.     //Configure DCMI interrupts
303.     dcmi->IER &= 0x000; //reset register
304.     //Enable line interrupt
305.     dcmi->IER |= (1<<4); //Yes, enable interrupt on line //FIX ME FIX ME good for debugging
306.     //Enable complete capture interrupt
307.     dcmi->IER |= 0x0001; //yes, enable interrupt on a whole frame
308.     //Enable overrun interrupt
309.     dcmi->IER |= (1<<1);
310.
311.
312.
313.     //Now set up DMA
314.     //Must use DMA2, STREAM_1, Channel 1 for DCMI (or DMA2, Stream 7, CH1)

315.     //p 310 of reference manual
316.     rcc_periph_clock_enable(RCC_DMA2);
317.
```

```
318.     //Disable to allow configuration
319.     dma_disable_stream(DMA2, DMA_STREAM1);
320.     //configure
321.     dma_stream_reset(DMA2, DMA_STREAM1);
322.
323.     //For DCMI, must use stream 1, channel 1
324.     dma_channel_select(DMA2, DMA_STREAM1, DMA_SxCR_CHSEL_1);
325.
326.     dma_set_transfer_mode(DMA2, DMA_STREAM1,
327.                           DMA_SxCR_DIR_PERIPHERAL_TO_MEM);
328.
329.     //Configure peripheral
330.     dma_set_peripheral_address(DMA2, DMA_STREAM1, (uint32_t)&dcmi->DR);
331.     dma_set_peripheral_burst(DMA2, DMA_STREAM1, DMA_SxCR_MBURST_SINGLE);
332.     //dma_set_peripheral_burst(DMA2, DMA_STREAM1, DMA_SxCR_MBURST_INCR4);
333.     dma_set_peripheral_size(DMA2, DMA_STREAM1, DMA_SxCR_PSIZE_32BIT);
334.     //dma_set_peripheral_size(DMA2, DMA_STREAM1, DMA_SxCR_PSIZE_16BIT);
335.     dma_disable_peripheral_increment_mode(DMA2, DMA_STREAM1);//don't want to increment addr
of data register
336.
337.     //Configure memory
338.     dma_enable_double_buffer_mode(DMA2, DMA_STREAM1);
339.     dma_set_memory_size(DMA2, DMA_STREAM1, DMA_SxCR_MSIZE_32BIT);//16BIT);
340.     dma_enable_memory_increment_mode(DMA2, DMA_STREAM1);
341.     dma_set_memory_burst(DMA2, DMA_STREAM1, DMA_SxCR_MBURST_SINGLE);
342.     //dma_set_memory_burst(DMA2, DMA_STREAM1, DMA_SxCR_MBURST_INCR4);
343.     //Set memory addresses
344.     dma_set_memory_address(DMA2, DMA_STREAM1, (uint32_t) &rawPicture);
345.     dma_set_memory_address_1(DMA2, DMA_STREAM1, (uint32_t) &rawPicture + BUFLENGTH);
346.     dma_set_initial_target(DMA2, DMA_STREAM1, 0);
347.
348.     //Other configuration
349.     dma_set_priority(DMA2, DMA_STREAM1, DMA_SxCR_PL_HIGH);
350.     //dma_enable_direct_mode(DMA2, DMA_STREAM1);
351.     dma_enable_fifo_mode(DMA2, DMA_STREAM1);
352.     dma_set_fifo_threshold(DMA2, DMA_STREAM1, DMA_SxFCR_FTH_4_4_FULL);
353.
354.
355.     dma_set_dma_flow_control(DMA2, DMA_STREAM1);
356.     dma_set_number_of_data(DMA2, DMA_STREAM1, BUFLENGTH);
357.
358.     //Enable all the other interrupts whilst testing //FIX ME FIX ME
359.     dma_enable_transfer_complete_interrupt(DMA2, DMA_STREAM1);
360.     dma_enable_transfer_error_interrupt(DMA2, DMA_STREAM1);
361.     //dma_enable_half_transfer_interrupt(DMA2, DMA_STREAM1);
362.     //dma_enable_direct_mode_error_interrupt(DMA2, DMA_STREAM1);
363.     dma_enable_fifo_error_interrupt(DMA2, DMA_STREAM1);
364.
365.
366.     //Enable DCMI interrupt
367.     nvic_enable_irq(NVIC_DCMI_IRQ);
368.     nvic_enable_irq(NVIC_DMA2_STREAM1_IRQ);
369.     //Want to set DCMI priority higher than DMA priority - memcpy takes much more time than
anything in the DCMI isr
370.     //FIX ME FIX ME - TEMPORARILY SWAP THEM TO SEE IF IT FIXES THE ISSUE OF DMA INTERRUPTS
NOT BEING CALLED OFTEN ENOUGH
371.     nvic_set_priority(NVIC_DCMI_IRQ, 52);
372.     nvic_set_priority(NVIC_DMA2_STREAM1_IRQ, 51);
373.
374.     //turn it on
375.     //dma_enable_stream(DMA2, DMA_STREAM1);
376.     dma_clear_interrupt_flags(DMA2, DMA_STREAM1, DMA_TCIF | DMA_DMEIF | DMA_FEIF | DMA_TEIF
| DMA_HTIF); //clear all interrupt flags
377.     //Turn on DCMI
378.     dcmi->CR |= (1 << 14); //enable DMA capture
379.     dcmi->CR |= 0b1;  //enable DCMI
380.
```

```
381.     }
382.
383.
384. void dcmi_isr(void){
385.     //DCMI interrupt
386.
387.     //The DCMI Peripheral is nice - there is a dedicated register that is the result of &&
the raw interrupt register (dcmi->RIS) with the interrupt enable register (dcmi->IER).
388.     //Hence don't need to check if interrupts are enabled
389.
390.     volatile DCMI_HW_Registers *dcmi = (DCMI_HW_Registers *) DCMI_BASE; //DCMI_BASE
provided by libopencm3
391.
392.
393.
394.     if ((dcmi->MIS & (1<<0)) != 0){
395.         //Frame interrupt
396.         //for (;;){};
397.         //row = 0;
398.         savePicture = 1;
399.         dma_enable_stream(DMA2, DMA_STREAM1); //turn DMA on
400.         //clear interrupt
401.         dcmi->ICR |= (1<<0);
402.     }
403.     if ((dcmi->MIS & (1<<4)) != 0){
404.         //Line interrupt
405.         //for (;;){};
406.         //row++;
407.         //Clear interrupt
408.         gpio_toggle(GPIOF, STATUSPIN2);
409.         dcmi->ICR |= (1<<4);
410.     }
411.     if ((dcmi->MIS & (1<<1)) != 0){
412.         //Data overrun flag
413.         //Clear interrupt
414.         dcmi->ICR |= (1<<1);
415.     }
416.
417.
418. }
419.
420.
421. void dma2_stream1_isr(void){
422.
423.     //check if  a transfer has been completed
424.     if (dma_get_interrupt_flag(DMA2, DMA_STREAM1, DMA_TCIF) != 0) {
425.         gpio_set(GPIOF, STATUSPIN);
426.         if (savePicture!=0){
427.             uint32_t *addr = (uint32_t *) &rawPicture;
428.             addr += (chunk+1)*BUFLENGTH/4; // /2 because uint16? // /4 if uint32?
429.             if (chunk <= MAXHEIGHT*MAXROWLENGTH/BUFLENGTH){ //#pixels / chunk size
430.                 switch (dma_get_target(DMA2, DMA_STREAM1)){
431.                     case 0:
432.                         //First memory target has the data => move target
433.                         dma_set_memory_address_1(DMA2, DMA_STREAM1, (uint32_t) addr);
434.                         //memcpy(addr+(rowLength*4*row), rowBuf2, 2*2*rowLength);
435.                         break;
436.                     case 1:
437.                         //Second memory target has the data => move the target
438.                         dma_set_memory_address(DMA2, DMA_STREAM1, (uint32_t) addr);
439.                         //memcpy(addr+(rowLength*4*row), rowBuf1, 2*2*rowLength);
440.                         break;
441.                 }
442.                 chunk++;
```

```
443.             } else{
444.                 savePicture = 0;
445.                 chunk = 0;
446.                 memcpy(&pictureBuf, &rawPicture, MAXHEIGHT*MAXROWLENGTH*2); //copyy the
picture to the picture Buffer
447.                 savePicture = 0; //Must be after memcpy to not copy half-picture
448.             }
449.         }
450.         dma_clear_interrupt_flags(DMA2, DMA_STREAM1, DMA_TCIF);//Clear interrupt flag to
avoid triggering too often
451.         gpio_clear(GPIOF, STATUSPIN);
452.     } else if (dma_get_interrupt_flag(DMA2, DMA_STREAM1, DMA_TEIF) !=0) {
453.         dma_clear_interrupt_flags(DMA2, DMA_STREAM1, DMA_TEIF);//Transfer error
454.     } else if (dma_get_interrupt_flag(DMA2, DMA_STREAM1, DMA_FEIF) != 0){
455.         dma_clear_interrupt_flags(DMA2, DMA_STREAM1, DMA_FEIF);//FIFO transfer error
456.     } else if (dma_get_interrupt_flag(DMA2, DMA_STREAM1, DMA_DMEIF) != 0){
457.         dma_clear_interrupt_flags(DMA2, DMA_STREAM1, DMA_DMEIF);//direct mode transfer
error
458.     }
459.
460.
461. }
```

I would greatly appreciate any help anyone can provide. I know this is possible - I keep seeing examples where it's been done - but I am somewhat stuck.