mico32_camera

October 24, 2011

1 Introduction

The *mico32_camera* is a wishbone compatible component and provides an interface between *LatticeMico32* microprocessor and a *CMOS camera HV7131GP*. The *mico32_camera* is configured in slave mode to acquire a video stream from the *CMOS camera,* and is later switched to master mode to transfer the stream to a memory device.

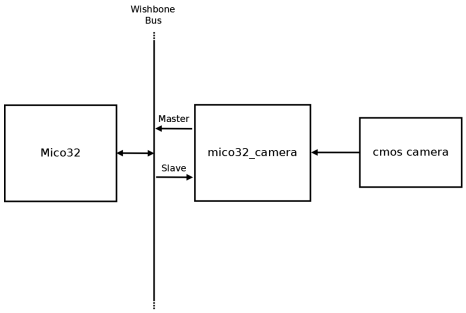

Figure 1: Using *mico32_camera* to communicate with a *CMOS camera HV7131GP*

2 Version

This document is version 1.003 of the *mico32_camera* component.

3 Features

The *mico32_camera* gets data from *CMOS Image Sensor HV7131GP* produced by *MagnaChip Semiconductor Ltd.* The *HV7131GP* datasheet is available on the internet.
The *mico32_camera* offers following features:

- *WISHBONE B.3* interface.

- Cofiguration in *master* and *slave* modes.
- Interrupt request to the processor can be done after image capture.
- Library of basic data structures and software routines for working with the controller.
- Configurable clock (*MCLK*) frequency for *CMOS Image Sensor*.
- Control of capture image size and resolution.
- Location of captured image storage in memory.

4 Functional Description

The I/O pins, timing cycle, and output signals of the *mico32_camera's CMOS Image Sensor* are described below.

4.1 *HV7131GP* output signals

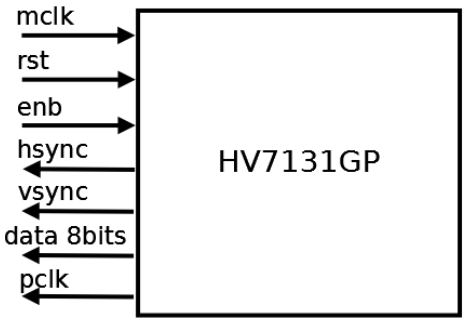

Figure 2: *hv7131gp* pins

- ***mclk***: The *hv7131gp* is essentially a synchronous logic circuit. Hence, it needs a signal clock to work.
- ***rst:*** Sensor Reset (for more information please refer *hv7131gp* datasheet).
- ***enb:*** *enb* low : sleep mode, *enb* high : normal mode. (for more information please refer *hv7131gp* datasheet).
- ***pclk:*** Video Output Clock
- ***hsync:*** Video *Horizontal* Line Synchronization signal. Image data is valid when *hsync* is high.
- ***vsync:*** Video *Frame* Synchronization signal. *vsync* is active at the start of the image data frame.

- ***data:*** Video Luminance *Data*[7:0]

Note: The *i2c* bus is not shown in the figure as it is managed by the microprocessor.

4.2 *HV7131GP* timing signals

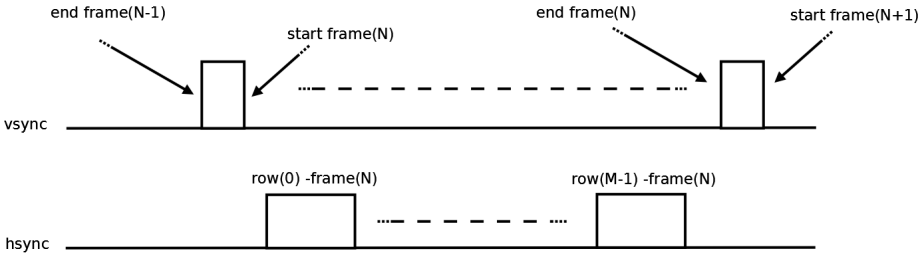

Figure 3: *vsync* and *hsync* timing

The *mico32_camera* polls for the vsync signal coming from the *CMOS Image Sensor hv3171gp*. After reset, on receiving the first sync signal, the component goes from RESET to IDLE state. Subsquently, the rising edge of the *vsync* signal is processed by the *mico32_camera* as end of the current frame. The falling edge of the *vsync* signal is processed by the *mico32_camera* as start of a new frame (Fig.3).

4.3 How *mico32_camera* works?

The *mico32_camera* can attain six different states (Fig.4). The current status of the component (*STATE_BITS*) is placed in an internal register (*STATE*) and can be read by the programmer anytime.

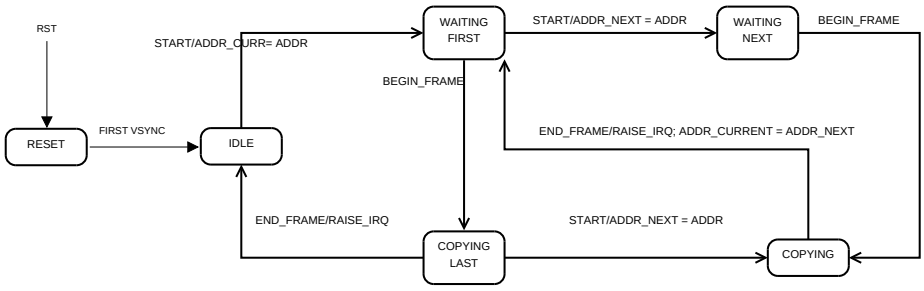

Figure 4: *mico32_camera* state diagram

Six different states of the component:

1. **Reset**: *STATE_BITS = 3'b000.* while in **Reset** state, the component sends an impulse signal to the ***hv7131gp*** sensor and the *vsync* signal is processed by the *mico32_camera* as end of the current frame. If the *CMOS image sensor* is not connected on the board, the *vsync* signal is not received and the **Reset** state remains unchanged.

2. **Idle**: *STATE_BITS = 3'b001.* The component is waiting for a *start command (CNTR_START*). On receving this *(CNTR_START* is asserted), the component goes from **Idle** state to **Waiting First** state. During this process, the address value is sampled and stored in the internal register *ADDR_CURR.*

3. **Waiting First**: *STATE_BITS = 3'b010.* The component is waiting for the start of a new frame (falling edge of the *vsync* signal). On receving this *(CNTR_START* is asserted), the component goes from**Waiting First** state to **Waiting Next** state. During this process, the address value is sampled and stored in the internal register *ADDR_NEXT.* This value read will be used for storing the next received frame. While the component is expecting a new command, the captured frame will get stored. On receving another new frame (falling edge of the *vsync* signal), the component goes to **Copying Last** state.

4. **Waiting Next**: *STATE_BITS = 3'b011.* The component is waiting for the start of a new frame (falling edge of the *vsync* signal). While the captured frame is being being stored in the memory, arrival of another new frame switches the component to the **Copying** state. Meanwhile, every another new start command *(CNTR_START* is asserted) received is ignored.

5. **Copying**: *STATE_BITS = 3b'100.* The component is writing the current frame into the memory and the next received frame is scheduled to be written in the memory. Once the writing process is completed (rising edge of the *vsync* signal), the component sets the *FLAG_IF* to "1" (refer sec. 9.3), and goes to **Waiting First** state. If the *IE_FLAG = 1'b1* (refer sec.9.5), it requests an interrupt to the *mico32* processor. Meanwhile, every another new start command *(CNTR_START* is asserted) received is ignored.

6. **Copying Last**: *STATE_BITS = 3'b101.* The component is writing the current frame into the memory, and this also happens to be the last frame being written. Once the writing process is completed (rising edge of the *vsync* signal), the component sets the *FLAG_IF* to "1", and goes to **Idle** state.When the writing finishes (rising *vsync* signal see Fig.3) the component sets *FLAG_IF* and switches to the **Idle** state. If the *IE_FLAG = 1'b1* (refer sec.9.5), it requests an interrupt to the *mico32* processor. Meanwhile, a new start command *(CNTR_START* is asserted) received, switches the component to **Copying** state. During this process, the ADDR value is sampled and will be used for the next frame.

Note: At any moment, the receipt of reset command (*CNTR_RST* is asserted) will switch the component from any given state to the **Reset** state.

5 Configuration

I/O port configuration, HDL (hardware description language) and GUI (graphical user interface) parameters for operating the *mico32_camera* are described below:

Field	***Note***	***Default***
Instance Name	Specifies the name of the *mico32_camera* instance.	*camera*
Base Address	Specifies the base address for the device. The minimum byte alignment is *0x80*.	*0x80000000*
Fifo Depth	Specifies the *Fifo* depth for storing data while wishbone bus is *busy*.	*16*

Table 1: GUI for *mico32_camera* configuration

6 I/O pins

The WISHBONE interface for operating the *mico32_camera* in **master** mode uses the *CTI_O* signal to boost cycle transfers. In this mode, at any given time, only fixed 4 bytes of data can be written on the memory bus. Hence, the *SEL_O* signal is fixed at *1111*. Also, cache line wrap is not supported in this mode. Hence, the *BTE_O* signal is fixed at *00*.
The *RTY_O* and *ERR_O* signals are not supported when the WISHBONE interface is operating the *mico32_camera* in **slave** mode. In the mode, the *INT_O* signal requests an interrupt to the *mico32* processor.
The I/O pin configuration for communicating with the *CMOS Image Sensor hv7131gp* are described below:

I/O Port	*Active*	*Direction*	*Initial State*	*Description*
mclk	*HIGH*	*O*	*X*	Input clock signal for *HV7131GP* camera sensor
rst	*HIGH*	*O*	*HIGH*	Sensor Reset
enb	*LOW*	*O*	*HIGH*	Signal Sleep Sensor
vsync	*HIGH*	*I*	*LOW*	Video Frame Synchronization signal. vsync is active at start of image data frame
hsync	*HIGH*	*I*	*LOW*	Video Horizontal Line Synchronization signal. Image data is valid, when *HSYNC* is high.
pclk	*HIGH*	*I*	*LOW*	Video Output Clock
y	*X*	*I*	*X*	Video Luminance Data[7:0]

Table 2: Pin for *HV7131GP* module

7 Device Driver

The *mico32_camera* driver functions are located in a header file *MicoCamera.h*. These are inline functions with macro values (of internal states of the component) and bit mask for registers.

- *Mico_camera_read_address*
- *Mico_camera_write_address*
- *Mico_camera_read_status_register*
- *Mico_camera_read_IF_flag*
- *Mico_camera_read_ERR_flag*
- *Mico_camera_start*
- *Mico_camera_clear_IRQ_flag*
- *Mico_camera_reset*
- *Mico_camera_enable_IRQ*
- *Mico_camera_disable_IRQ*
- *Mico_camera_IRQ_enabled*
- *Mico_camera_write_divisor*
- *Mico_camera_read_divisor*
- *Mico_camera_frame_terminated*

- *Mico_camera_size_value*
- *Mico_camera_size_frame_error*

8 *mico32_camera* registers

When the *mico32_camera* is connected to the Wishbone bus, the following registers are available from *BASE_ADDRESS* component:

Name Register	***Offset***	***Num. Bits***	***Type***
ADDR	*0x00*	*32*	*R/W*
STATE	*0x04*	*32*	*R*
FLAG	*0x08*	*32*	*R*
CNTR1	*0x0C*	*32*	*W*
CNTR2	*0x10*	*32*	*R/W*
DIV	*0x14*	*32*	*R/W*
COUNT	*0x18*	*32*	*R/W*

8.1 Read a *mico32_camera* register

The registers *ADDR*, *STATE*, *FLAG*, *CNTR2*, *DIV* and *COUNT* permit their values to be read. The following program shows, for example, how to read the *COUNT* register:

```
1
2  /* Offsets mico32_camera resister from address_base */
3  #define ADDR_OFF     0x00000000
4  #define STATE_OFF    0x00000004
5  #define FLAG_OFF     0x00000008
6  #define CNTR2_OFF    0x00000010
7  #define DIV_OFF      0x00000014
8  #define COUNT_OFF    0x00000018
9
10 int main(void) {
11
12       /* Macro CAMERA_BASE_ADDRESS is placed in system_conf.h file */
13       unsigned long *register = (CAMERA_BASE_ADDRESS + COUNT_OFF);
14       unsigned long count_value;
15
16       ...
17       /* Read count register. */
18       count_value = *register;
19
20       ...
21       return 0;
22 }
```

8.2 Write in a mico32_camera register

The registers *ADDR*, *CNTR1*, *CNTR2, DIV* and *COUNT* permit their values to be written. The follow program shows how to write the *ADDR* register:

```
1
2   /* Value to write in ADDR register  */
3   #define ADDR_VALUE  0x00010000
4
5   /* Offsets mico32_camera resister from address_base */
6   #define ADDR_OFF     0x00000000
7   #define STATE_OFF    0x00000004
8   #define FLAG_OFF     0x00000008
9   #define CNTR2_OFF    0x00000010
10  #define DIV_OFF      0x00000014
11  #define COUNT_OFF    0x00000018
12
13  int main(void) {
14
15        /* Macro CAMERA_BASE_ADDRESS is placed in system_conf.h file */
16        unsigned long *reg_addr = (CAMERA_BASE_ADDRESS + ADDR_OFF);
17
18        ...
19        /* Write in the addr register. */
20        *reg_addr = ADDR_VALUE;
21
22        ...
23        return 0;
24  }
```

9 *mico32_camera's* registers

As mentioned in Sec. 8, the *mico32_camera* has seven registers. These registers are described below in detail:

9.1 *ADDR* register (dim: *32 bits*, type: *R/W*, offset: *0x00000000*)

Before acquiring a new frame, the *mico32_camera* should know the first valid storage location address. For all subsquent frame acquisitions, while the current frame is being saved, the *mico32_camera* increments the memory pointer to the appropriate storage location address.

ADDR31	*ADDR30*	.....	*ADDR0*

Note: The value written in the *ADDR* register must point to a valid memory address. The *ADDR* pointing to an invalid memory area could corrupt the program list code in the memory!!!
The following program shows how to write a valid storage location address value in the *ADDR* register:

```
1
2   /*
3      SDRAM device is reserved only for data.
4      It is mapped on WishBone from: 0x04000000 to 0x05FFFFFF
5   */
6
7   /*
8      Value to write in ADDR register
```

```
9       We want write the data image in sdram memory
10   */
11   #define ADDR_VALUE  0x04010000
12
13   /* Image 640 x 480 1 byte color
14      Image size in memory is: 640*480*1/4 (The
15      mico32_camera write 4 byte for time in memory)
16   */
17   #define IMAGE_SIZE  640*480*1/4
18
19   /* Offsets mico32_camera resister from address_base */
20   #define ADDR_OFF     0x00000000
21   #define STATE_OFF    0x00000004
22   #define FLAG_OFF     0x00000008
23   #define CNTR2_OFF    0x00000010
24   #define DIV_OFF      0x00000014
25   #define COUNT_OFF    0x00000018
26
27   int main(void) {
28
29        /* Macro CAMERA_BASE_ADDRESS is placed system_conf.h file */
30        unsigned long *reg_addr = (CAMERA_BASE_ADDRESS + ADDR_OFF);
31
32        ...
33        /*
34           SDRAM_SIZE > (ADDR_VALUE + IMAGE_SIZE)
35        */
36        *reg_addr= ADDR_VALUE;
37
38        ...
39        return 0;
40   }
```

9.2 *STATE* register (dim: *32 bits*, type: *R*, offset: *0x00000004*)

The *STATE* register stores all the nternal state information of the *mico32_camera*.

Bit31	*Bit30*	.....	*Bit3*	*STATE_S2*	*STATE_S1*	*STATE_S0*

9.2.1 *STATE_S* (dim: *3 bits*, type: *R*, offset: *0x00*, reset value: *000*)

The internal states are encoded with three least significant bits (*STATE_S*); while other bits always fixed to "0" value:

- 000 => ***Reset***
- 001 => ***Idle***
- 010 => ***Waiting First***
- 011 => ***Waiting Next***
- 100 => ***Copying***
- 101 => ***Copying Last***

The following program shows polling done by the *STATE* register until the *mico32_camera* goes into *IDLE* state:

```
1
2   /* Offsets mico32_camera resister from address_base */
3   #define ADDR_OFF      0x00000000
4   #define STATE_OFF     0x00000004
5   #define FLAG_OFF      0x00000008
6   #define CNTR2_OFF     0x00000010
7   #define DIV_OFF       0x00000014
8   #define COUNT_OFF     0x00000018
9
10  /* mico32_camera states */
11  #define RESET           0
12  #define IDLE            1
13  #define WAITING_FIRST   2
14  #define WAITING_NEXT    3
15  #define COPYING         4
16  #define COPYING_LAST    5
17
18  int main(void) {
19
20      /* Macro CAMERA_BASE_ADDRESS is defined in system_conf.h file */
21      unsigned long *reg_state = (CAMERA_BASE_ADDRESS + STATE_OFF);
22      unsigned long state;
23
24      ...
25      /* Waiting IDLE mico32_camera state */
26      do {
27            state = *reg_state;
28            ...
29
30      } while (state != IDLE);
31
32      ...
33      return 0;
34  }
```

9.3 *FLAG* register (dim: *32 bits*, type: *R*, offset: *0x00000008*)

The **FLAG** register is composed of three different bits:

Bit31	*Bit30*	.....	*SIZE_ERR*	*ERR_FLAG*	*IF_FLAG*

9.3.1 *IF_FLAG* (dim: *1 bit*, type: *R*, mask: *0x00000001*, reset value: 0).

This bit switches from "0" to "1" at end of an frame acquisition.

- From 0 to 1 => A frame has been completely written, and the *mico32_camera* has either switched from **Writing Last** state to **Idle** state or from **Writing** state to **Waiting Next** state.
- 0 => No frame has been acquired since the last assertion of *IF_RESET*.

- An interrupt to the *Mico32* processor is raised if the programmer has set the interrupt enable bit *IE_FLAG = 1'b1*. Else, after finishing with all frame acquisitions, the *IF_RESET* should set the *IF_FLAG* value to "0".

The following program shows polling done by the *IF_FLAG* register to check the end of frame capture i.e. the rising edge of the *vsync* signal:

```
1
2  /* Mask of the IF_FLAG */
3  #define MASK_IF_FLAG   0x00000001
4
5  /* Offsets mico32_camera resister from address_base */
6  #define ADDR_OFF       0x00000000
7  #define STATE_OFF      0x00000004
8  #define FLAG_OFF       0x00000008
9  #define CNTR2_OFF      0x00000010
10 #define DIV_OFF        0x00000014
11 #define COUNT_OFF      0x00000018
12
13 int main(void) {
14
15       /* Macro CAMERA_BASE_ADDRESS is defined system_conf.h file */
16       unsigned long *reg_flag = (CAMERA_BASE_ADDRESS + FLAG_OFF);
17       unsigned long bit_if_flag;
18
19       ...
20       /* Start capture image */
21
22       ...
23       /* Read bit IF_FLAG for end capture */
24       do {
25             ...
26             bit_if_flag = *reg_flag & MASK_IF_FLAG;
27
28             ...
29       } while (!bit_if_flag);
30
31       ...
32       return 0;
33 }
```

9.3.2 *ERR_FLAG* (dim: *1 bit*, type *R*, mask: *0x00000002*, reset: *0*).

This flag is set to "1" if during a bus cycle access any Wishbone error is encountered or there is an overflow in the FIFO **Master**. This flag will be reset to "0" when a new frame is being acquired.
The following program checks the value of *ERR_FLAG* post frame acquisition:

```
1
2  /* Mask of the ERR_FLAG */
3  #define MASK_FLAG_ERR 0x00000002
4
5  /* Size image: 640x480x2 colors (2 byte).
6  #define IMAGE_SIZE     (640*480*2)
```

```
7
8   /* Offsets mico32_camera resister from address_base */
9   #define ADDR_OFF        0x00000000
10  #define STATE_OFF       0x00000004
11  #define FLAG_OFF        0x00000008
12  #define CNTR2_OFF       0x00000010
13  #define DIV_OFF         0x00000014
14  #define COUNT_OFF       0x00000018
15
16  int main(void) {
17
18        /* Macro CAMERA_BASE_ADDRESS from system_conf.h */
19        unsigned long *reg_flag = (CAMERA_BASE_ADDRESS + FLAG_OFF);
20        unsigned long bit_err_flag;
21
22        ...
23        /* Start capture image */
24
25        ...
26        /* Read bit ERR_FLAG */
27        bit_err_flag = *reg_flag & MASK_ERR_FLAG;
28
29        if (bit_err_flag) {
30
31              /* There was a error on wishbone and/or overflow data fifo */
32
33        } else {
34
35              /* There was not error */
36
37        }
38
39        ...
40        return 0;
41  }
```

9.3.3 SIZE_ERR (dim: *1 bit*, type *R*, mask: *0x00000004*, reset: *0*).

This bit is set to "1" if the frame size (in bytes) stored in the memory is different from the value specified by the user in the *COUNT* register. This bit assumes "0" value when above condtion is not met.
The following program checks the value of *SIZE_ERR* after writing the acquired frame:

```
1
2   /* Mask of the SIZE_ERR */
3   #define MASK_SIZE_ERR 0x00000004
4
5   /* Size image: 640x480 colors (2 byte). */
6   #define IMAGE_SIZE     ((640*480*2)/4)UL
7
8   /* Offsets mico32_camera register from address_base */
9   #define ADDR_OFF        0x00000000
10  #define STATE_OFF       0x00000004
11  #define FLAG_OFF        0x00000008
12  #define CNTR2_OFF       0x00000010
```

```
13  #define DIV_OFF         0x00000014
14  #define COUNT_OFF       0x00000018
15
16  int main(void) {
17
18        /* Macro CAMERA_BASE_ADDRESS from system_conf.h */
19        unsigned long *reg_count = (CAMERA_BASE_ADDRESS + COUNT_OFF);
20        unsigned long *reg_flag = (CAMERA_BASE_ADDRESS + FLAG_OFF);
21        unsigned long bit_size_err;
22
23        ...
24        /* With a write in count register we enable the image size control. */
25        *reg_count = IMAGE_SIZE;
26
27        ...
28        /* Capture an image */
29
30        ...
31        /* Read bit SIZE_ERR */
32        bit_size_err = *reg_flag & MASK_SIZE_ERR;
33
34        if (bit_size_err) {
35
36              /* There was a size error */
37
38        } else {
39
40              /* There was not size error */
41
42        }
43
44        ...
45        return 0;
46  }
```

9.4 *CNTR1* register (dim: *32 bits*, type: *W*, offset: *0x0000000C*)

This register is composed of three functional bits:

Bit31	*Bit30*	.....	*CNTR_RST*	*IF_RESET*	*CNTR_S*

9.4.1 *CNTR_START* (dim: *1 bit*, type: *W*, mask: *0x00000001*, reset: -)

When this bit is set to "1", the *mico32_camera* starts to acquire a new frame. Note: Before acquiring a new image, ensure that the *ADDR* points to a valid memory area. The *ADDR* pointing to an invalid memory area could corrupt the program list code in the memory!!!
The following program writes a valid address value in the *ADDR* register and subsquently starts to acquire image.

```
1
2   /* SDRAM is mapped on WishBone from: 0x04000000 to 0x05FFFFFF */
3
4   /* Value to write in ADDR register
```

```
5       We want write the data image in sdram memory */
6   #define ADDR_VALUE  0x04010000
7
8   /* Mask of the CNTR_START */
9   #define MASK_CNTR_START  0x00000001
10
11  /* Offsets mico32_camera resister from address_base */
12  #define ADDR_OFF         0x00000000
13  #define STATE_OFF        0x00000004
14  #define FLAG_OFF         0x00000008
15  #define CNTR1_OFF        0x0000000C
16  #define CNTR2_OFF        0x00000010
17  #define DIV_OFF          0x00000014
18  #define COUNT_OFF        0x00000018
19
20  int main(void) {
21
22        /* Macro CAMERA_BASE_ADDRESS from system_conf.h */
23        unsigned long *reg_addr  = (CAMERA_BASE_ADDRESS + ADDR_OFF);
24        unsigned long *reg_cntr1 = (CAMERA_BASE_ADDRESS + CNTR1_OFF);
25        unsigned long *reg_flag  = (CAMERA_BASE_ADDRESS + FLAG_OFF);
26
27        ...
28        /* The ADDR register must points a valid memory area */
29        *reg_addr = ADDR_VALUE;
30
31        /* Start capture image */
32        *reg_cntr1 = MASK_CNTR_START;
33
34        ...
35        /* Read bit IF_FLAG for end capture */
36        do {
37              ...
38              /* Polling bit IF_FLAG from FLAg register.
39              bit_if_flag = *reg_flag & MASK_IF_FLAG;
40              ...
41
42        } while (!bit_if_flag);
43
44        ...
45        return 0;
46  }
```

9.4.2 IF_RESET (dim: *1 bit*, type: *W*, mask: *0x00000002*, reset: -)

When this bit is set to "1", the *IF_FLAG* is reset to "0".
The following program shows polling done by the *IF_FLAG* register to check the end of frame capture i.e. the rising edge of the *vsync* signal, and subsquently the *IF-FLAG* register will be reset to "0":

```
1
2   /* Number capture image */
3   #define NUM_CAP_IMAG  3
4
5   /* Mask of the IF_FLAG */
6   #define MASK_IF_FLAG  0x00000001
```

```
7
8   /* Mask of the IF_RESET */
9   #define MASK_IF_RESET 0x00000002
10
11  /* Offsets mico32_camera resister from address_base */
12  #define ADDR_OFF        0x00000000
13  #define STATE_OFF       0x00000004
14  #define FLAG_OFF        0x00000008
15  #define CNTR1_OFF       0x0000000C
16  #define CNTR2_OFF       0x00000010
17  #define DIV_OFF         0x00000014
18  #define COUNT_OFF       0x00000018
19
20  int main(void) {
21
22        /* Macro CAMERA_BASE_ADDRESS from system_conf.h */
23        unsigned long *reg_flag = (CAMERA_BASE_ADDRESS + FLAG_OFF);
24        unsigned long *reg_cntr1 = (CAMERA_BASE_ADDRESS + CNTR1_OFF);
25        unsigned long bit_if_flag;
26        unsigned int  i;
27
28        ...
29        /* Start capture image */
30
31        ...
32        i = 0;
33        /* Read bit IF_FLAG for end capture */
34        do {
35                ...
36                /* Start capture image */
37                *reg_cntr1 = MASK_CNTR_START;
38
39                do{
40
41                        /* Read if_flag from FRAG register */
42                        bit_if_flag = *reg_flag & MASK_IF_FLAG;
43
44                } while (!bit_if_flag);
45
46                /* Clear IF_RESET by program, necessary for next capture */
47                *reg_cntr1 = 0;
48                ...
49
50        } while ( ++i < NUM_CAP_IMAG);
51
52        ...
53        return 0;
54  }
```

9.4.3 CNTR_RST (dim: *1 bit*, type: *W*, mask: *0x00000004*, reset: -)

When this bit is set to "1", the component switches to **Reset** state.
The following program shows resetting of the *Mico32_camera*:

```
1
```

```
2    /* Mask of the CNTR_RST bit */
3   #define MASK_CNTR_RST     0x00000004
4
5   /* Offsets mico32_camera resister from address_base */
6   #define ADDR_OFF          0x00000000
7   #define STATE_OFF         0x00000004
8   #define FLAG_OFF          0x00000008
9   #define CNTR1_OFF         0x0000000C
10  #define CNTR2_OFF         0x00000010
11  #define DIV_OFF           0x00000014
12  #define COUNT_OFF         0x00000018
13
14  int main(void) {
15        /* Macro CAMERA_BASE_ADDRESS from system_conf.h */
16        unsigned long *reg_state = (CAMERA_BASE_ADDRESS + STATE_OFF);
17        unsigned long *reg_cntr1 = (CAMERA_BASE_ADDRESS + CNTR1_OFF);
18        unsigned lond state;
19
20        /* Reset mico32_camera */
21        *reg_cntr1 = MASK_CNTR_RST;
22
23        /* Waiting IDLE mico32_camera state */
24        do {
25
26              state = *register_state;
27              ...
28
29        } while (state != IDLE);
30
31         ...
32        return 0;
33  }
```

9.5 CNTR2 register (dim: 32 bits, type: R/W, offset: 0x10)

9.5.1 *IE_FLAG* (dim: *1 bit,* type: *R/W*, mask: *0x00000001*, reset: 0).

This bit is a mask to IF_FLAG.

Bit31	*Bit30*	.....	*Bit1*	*IE_FLAG*

- 1 => Interrupts are enabled.
- 0 => Interrupts are disabled.

The following program shows enabling/disabling of interrupts:

```
1
2   #define MASK_IE_FLAG      0x00000001
3   #define MASK_CNTR_START   0x00000001
4   #define MASK_IF_RESET     0x00000002
5
6   /* Offsets mico32_camera resister from address_base */
7   #define ADDR_OFF          0x00000000
```

```
 8  #define STATE_OFF          0x00000004
 9  #define FLAG_OFF           0x00000008
10  #define CNTR1_OFF          0x0000000C
11  #define CNTR2_OFF          0x00000010
12  #define DIV_OFF            0x00000014
13  #define COUNT_OFF          0x00000018
14
15  /* Macro CAMERA_BASE_ADDRESS from system_conf.h */
16  unsigned long *reg_cntr1 = (CAMERA_BASE_ADDRESS + CNTR1_OFF);
17  unsigned long *reg_cntr2 = (CAMERA_BASE_ADDRESS + CNTR2_OFF);
18
19  /* Prototype Interrupt handler end image acquisition */
20  void frame_isr(void);
21
22  int main(void) {
23       /*
24          Record handler camera interrupt
25          See: LatticeMico32 Software Developer User Guide,
26          on Lattice Semiconductor Web site
27       */
28
29       /*
30          Enable Mico32 external interrupt
31          (in this case from mico32_camera).
32       */
33       MicoEnableInterrupt(mico32_camera_irq);
34
35       /*
36          Record handler camera interrupt
37          See: LatticeMico32 Software Developer User Guide,
38          on Lattice Semiconductor Web site
39       */
40       MicoRegisterISR(mico32_camera_irq, NULL, frame_isr);
41
42       /* mico32_camera is enabled to send an interrupt */
43       *reg_cntr2 = MASK_IE_FLAG;
44       ...
45
46       /* Start capture image */
47       *reg_cntr1 = MASK_CNTR_START;
48       ...
49
50       return 0;
51  }
52
53  /* Interrupt handler end image acquisition */
54  void frame_isr(void) {
55       ...
56
57       /* Acknowledge interrupt */
58       *reg_cntr1 = MASK_IF_RESET;
59
60       ...
61  }
```

9.6 DIV register (dim: 32 bits, type: R/W, offset: 0x14)

This register is used for scaling the frequency of master clock (mclk) for the *CMOS Image Sensor hv3171gp.*

Bit31	***.....***	***Bit8***	***DIV_V7***	***.....***	***DIV_V1***	***DIV_V0***

9.6.1 DIV_VALUE (dim: 8 bits, type: R/W, reset: 0).

The value of the mclk is obtained by dividing the w_clk (clock Wishbone) by $2 * (div\_value + 1)$ as shown below:

DIV_VALUE	w_clk(MHz)	mclk(Mhz)
0	50	$50/(2 * (div\_value + 1)) = 25Mhz$
1	50	$50/(2 * (div\_value + 1)) = 12.5Mhz$
2	50	$50/(2 * (div\_value + 1)) = 8.333Mhz$
3	50	$50/(2 * (div\_value + 1)) = 6.25Mhz$

The following program shows how to write in the *DIV_VALUE* register:

```
1
2  /* Offsets mico32_camera resister from address_base */
3  #define ADDR_OFF          0x00000000
4  #define STATE_OFF         0x00000004
5  #define FLAG_OFF          0x00000008
6  #define CNTR1_OFF         0x0000000C
7  #define CNTR2_OFF         0x00000010
8  #define DIV_OFF           0x00000014
9  #define COUNT_OFF         0x00000018
10
11 /* Macro CAMERA_BASE_ADDRESS from system_conf.h */
12 unsigned long *reg_div = (CAMERA_BASE_ADDRESS + DIV_OFF);
13
14 int main(void) {
15
16       ...
17       /*
18          With w_clk = 50MHz and div value 3,
19          we have a mclk = 50/(2*(3+1))= 6.25MHz.
20       */
21       *reg_div = MASK_CNTR_START;
22
23       ...
24       return 0;
25 }
```

9.7 *COUNT* register (dim: 32 bits, type: R/W, offset: 0x18)

This register should be set to the size value of the frame in word (4 bytes) to be stored in the memory. When checking of the frame size is not required, set this value to "0".

COUNT31	.....	*COUNT7*	.....	*COUNT0*

The following program shows how to write the frame size in word (4 bytes) to be stored in the memory in the *COUNT* register:

```
1
2   /* Image size in byte: 640 x 480, 1 byte data. */
3   #define IMAGE_SIZE_WORD  ((680*480*1)/4)
4
5   /* Offsets mico32_camera resister from address_base */
6   #define ADDR_OFF          0x00000000
7   #define STATE_OFF         0x00000004
8   #define FLAG_OFF          0x00000008
9   #define CNTR1_OFF         0x0000000C
10  #define CNTR2_OFF         0x00000010
11  #define DIV_OFF           0x00000014
12  #define COUNT_OFF         0x00000018
13
14  /* Macro CAMERA_BASE_ADDRESS from system_conf.h */
15  unsigned long *reg_count = (CAMERA_BASE_ADDRESS + COUNT_OFF);
16  unsigned long *reg_flag = (CAMERA_BASE_ADDRESS + FLAG_OFF);
17
18  int main(void) {
19
20        ...
21        /*
22           COUNT register the component is enabled to
23           size control is on.
24        */
25        *reg_count = IMAGE_SIZE;
26
27        ...
28        /* Start capture image */
29        *reg_cntr1 = MASK_CNTR_START;
30
31        ...
32        /* End capture image. */
33
34        /* Read bit SIZE_ERR */
35        bit_size_err = *reg_flag & MASK_SIZE_ERR;
36
37        if (bit_size_err) {
38
39              ...
40              /* There was a size error */
41
42        } else {
43
44              ...
45              /* There was not size error */
46
47        }
48
49        ...
50        return 0;
51  }
52
53  /* Interrupt handler end image acquisition */
```

```
54  void frame_isr(void) {
55
56      ...
57      /* Acknowledge interrupt */
58      *reg_cntr1 = MASK_IF_RESET;
59
60      ...
61  }
```