HAN AEA - Embedded Vision & Machine Learning

EVD1 - Week 1

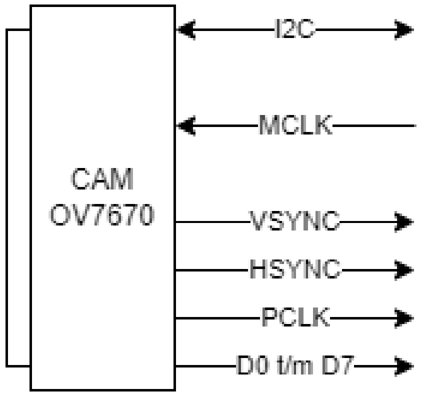
OV7670
SmartDMA
UVC

By Hugo Arends

HAN_UNIVERSITY OF APPLIED SCIENCES

OV7670

- There are several camera interfaces, such as
  - MIPI CSI (Mobile Industry Processor Interface) (Camera Serial Interface)
  - DVP (Digital Video Port)
- DVP is a parallel interface consisting of the following connections:
  - Data line (D[0:7])
  - Horizontal Sync (HSYNC)
  - Vertical Sync (VSYNC)
  - Pixel Clock (PCLK)
- The OV7670 camera module implements the DVP interface, and on top of that:
  - Input clock (MCLK)
  - I2C (SDA/SCL)

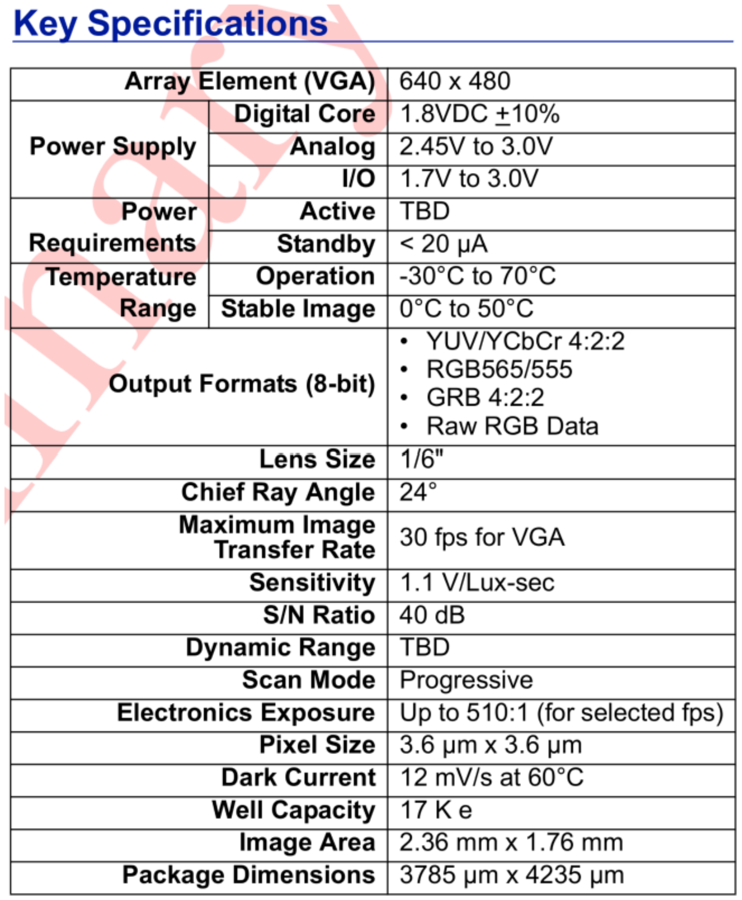


HAN_UNIVERSITY OF APPLIED SCIENCES

OV7670

- The OV760 supports several output formats, such as:
  - YUV422
  - RGB565
  - RGB888 (a.k.a. Raw RGB Data)
- The fsl_ov7670 driver, however, only supports the following 16-bit output formats:
  - YUYV
  - RGB565
  - RGBX4444
  - XRGB4444
  - XRGB1555

See the function OV7670_Init() In the file fsl_ov7670.c



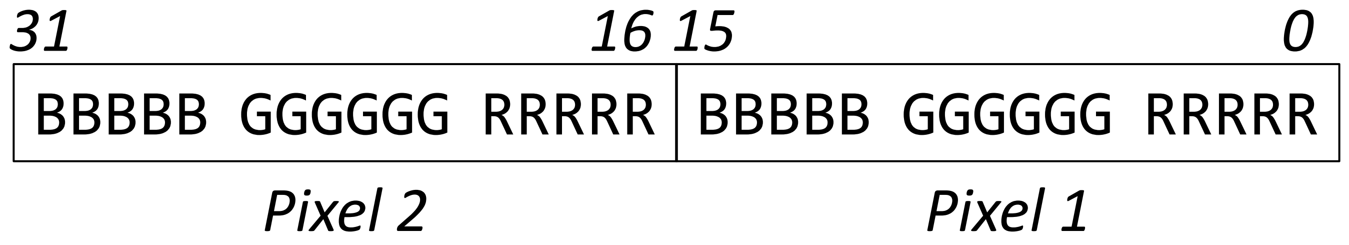
Key Specifications

Array Element (VGA)		640 x 480
Power Supply	Digital Core	1.8VDC ±10%
	Analog	2.45V to 3.0V
	I/O	1.7V to 3.0V
Power Requirements	Active	TBD
	Standby	< 20 μA
Temperature Range	Operation	-30°C to 70°C
	Stable Image	0°C to 50°C
Output Formats (8-bit)		• YUV/YCbCr 4:2:2 • RGB565/555 • GRB 4:2:2 • Raw RGB Data
Lens Size		1/6"
Chief Ray Angle		24°
Maximum Image Transfer Rate		30 fps for VGA
Sensitivity		1.1 V/Lux-sec
S/N Ratio		40 dB
Dynamic Range		TBD
Scan Mode		Progressive
Electronics Exposure		Up to 510:1 (for selected fps)
Pixel Size		3.6 μm x 3.6 μm
Dark Current		12 mV/s at 60°C
Well Capacity		17 K e
Image Area		2.36 mm x 1.76 mm
Package Dimensions		3785 μm x 4235 μm

OV7670

Knowing that there is support for 16-bit data formats only, which format to choose?

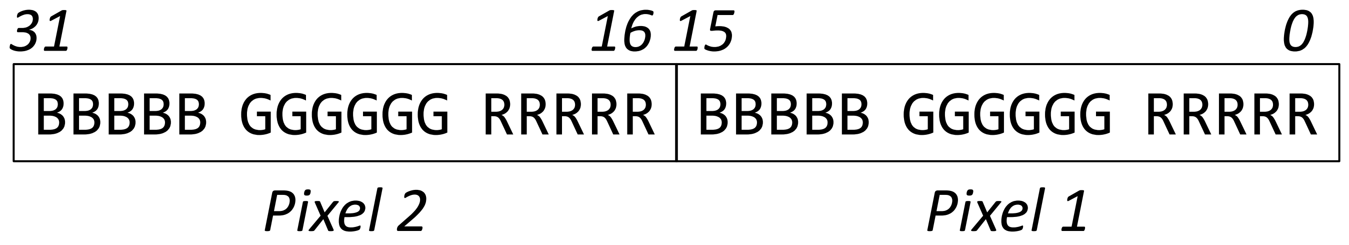
- RGB565 has distinct colour information for each pixel, but only a limited range per color channel



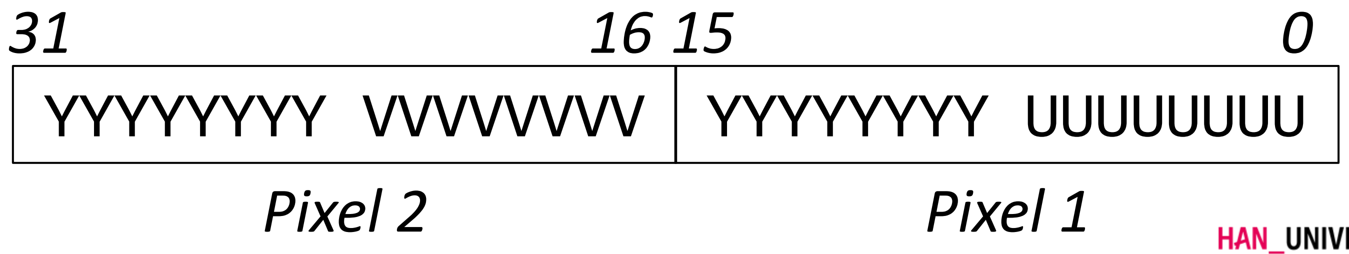
OV7670

Knowing that there is support for 16-bit data formats only, which format to choose?

- RGB565 has distinct colour information for each pixel, but only a limited range per color channel



- YUV422 has a full 8-bit range for intensity (Y), but two pixels share the same chroma values (U and V channels)



HAN_UNIVERSITY OF APPLIED SCIENCES

OV7670

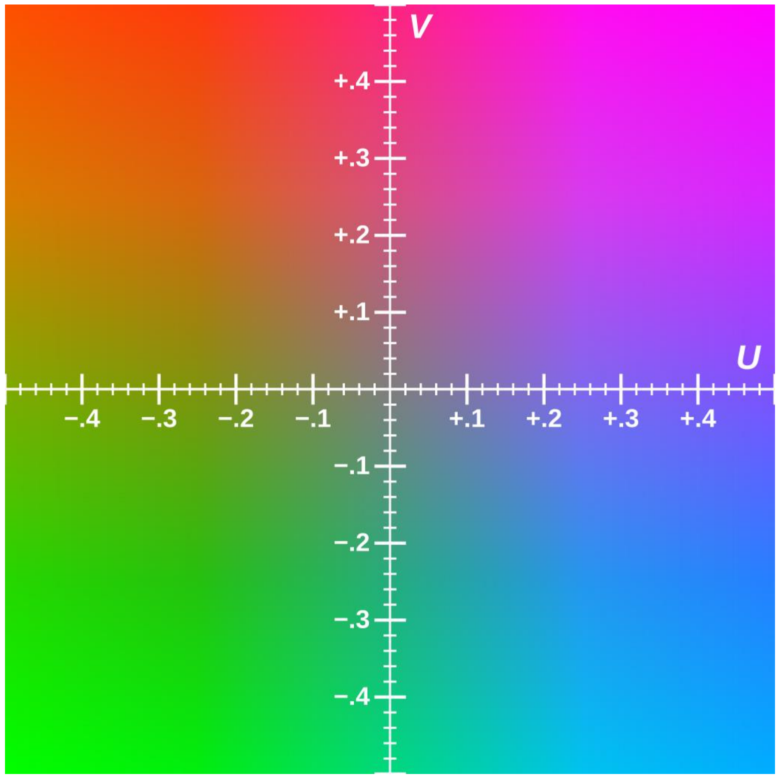
- The human eye is more sensitive to brightness information. Full scale brightness information is transmitted for every pixel in YUV422. In other words, only chrominance (color) information is discarded to achieve compression
- Conversion between 8-bit grayscale and YUV is easy, because the Y channel holds the 8-bit grayscale information
- Both RGB565 and YUV422 are supported UVC video formats

Conclusion: Select YUV422

YUV

- YUV pixels were invented when engineers wanted color television in a black-and-white infrastructure
- Y is called the *luminance* value
- U and V are *color difference* values
  - The lower (-0.5) or higher (+0.5) the values are, the more saturated (colorful) the pixel gets
  - For uint8 this translates to
    - `-0.5 =   0 (0x00)`
    - ` 0.0 = 128 (0x80)`
    - ` 0.5 = 255 (0xFF)`
- YUV comes in different subsampling schemes, such as YUV422, YUV411 and YUV444

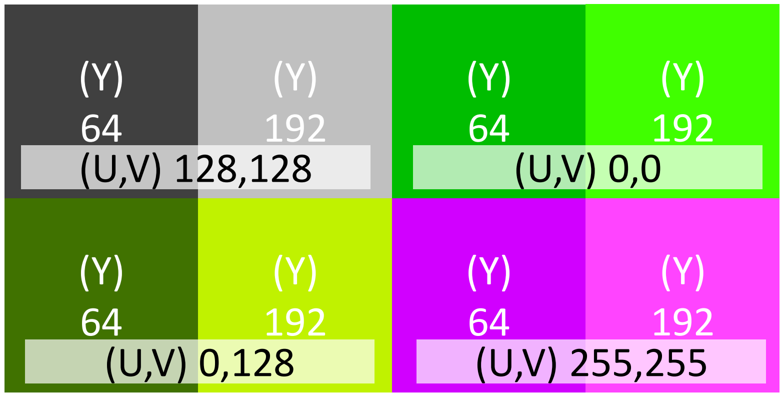
U-V color plane



HAN_UNIVERSITY OF APPLIED SCIENCES

YUV

Examples



HAN_UNIVERSITY OF APPLIED SCIENCES

YUV

YUV422 byte order storage formats

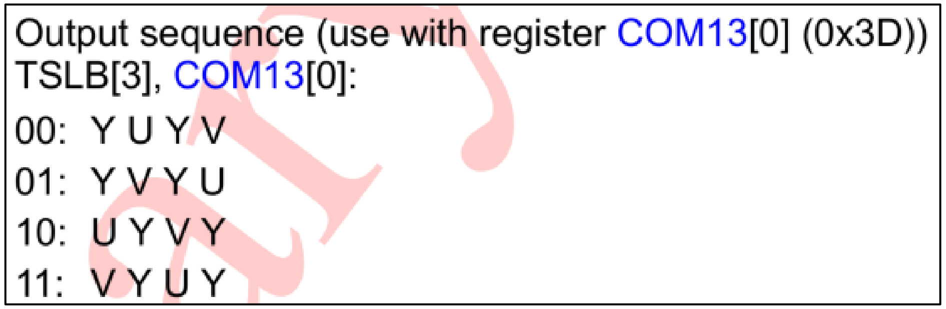
	31	*16*	*15*	*0*
Y U Y V	VVVVVVVV	YYYYYYYY	UUUUUUUU	YYYYYYYY
Y V Y U	UUUUUUUU	YYYYYYYY	VVVVVVVV	YYYYYYYY
U Y V Y	YYYYYYYY	VVVVVVVV	YYYYYYYY	UUUUUUUU
V Y U Y	YYYYYYYY	UUUUUUUU	YYYYYYYY	VVVVVVVV
	Pixel 2		*Pixel 1*	

HAN_UNIVERSITY OF APPLIED SCIENCES

YUV

YUV422 byte order storage formats

Can be configured in the camera with the bits TSLB[3] and COM13[0]



Output sequence (use with register COM13[0] (0x3D))
TSLB[3], COM13[0]:
00: Y U Y V
01: Y V Y U
10: U Y V Y
11: V Y U Y

After Initialisation with the provided driver function:

```
TSLB:  0x08 = 0b00001000
COM13: 0x88 = 0b10001000
```

Conclusion: U Y V Y

YUV

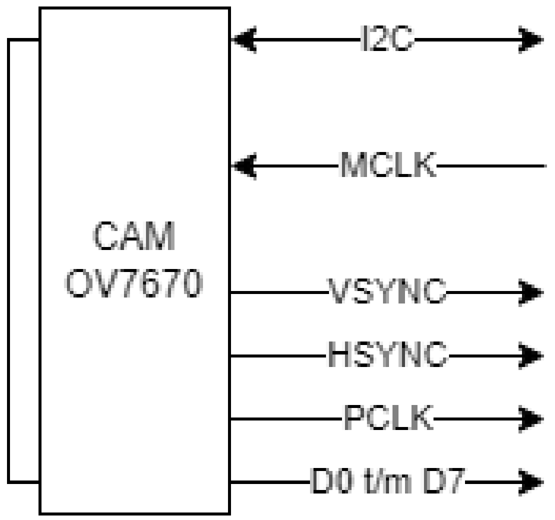
Notes

```
/// \brief Type definition of an uyvy pixel
///
/// 32 bits per two pixels stored in the following format:
///
///  31                               0
/// |YYYYYYYY VVVVVVVV|YYYYYYYY UUUUUUUU|
/// |     pixel 2     |     pixel 1     |
typedef uint16_t uyvy_pixel_t;
```

HAN_UNIVERSITY OF APPLIED SCIENCES

SmartDMA

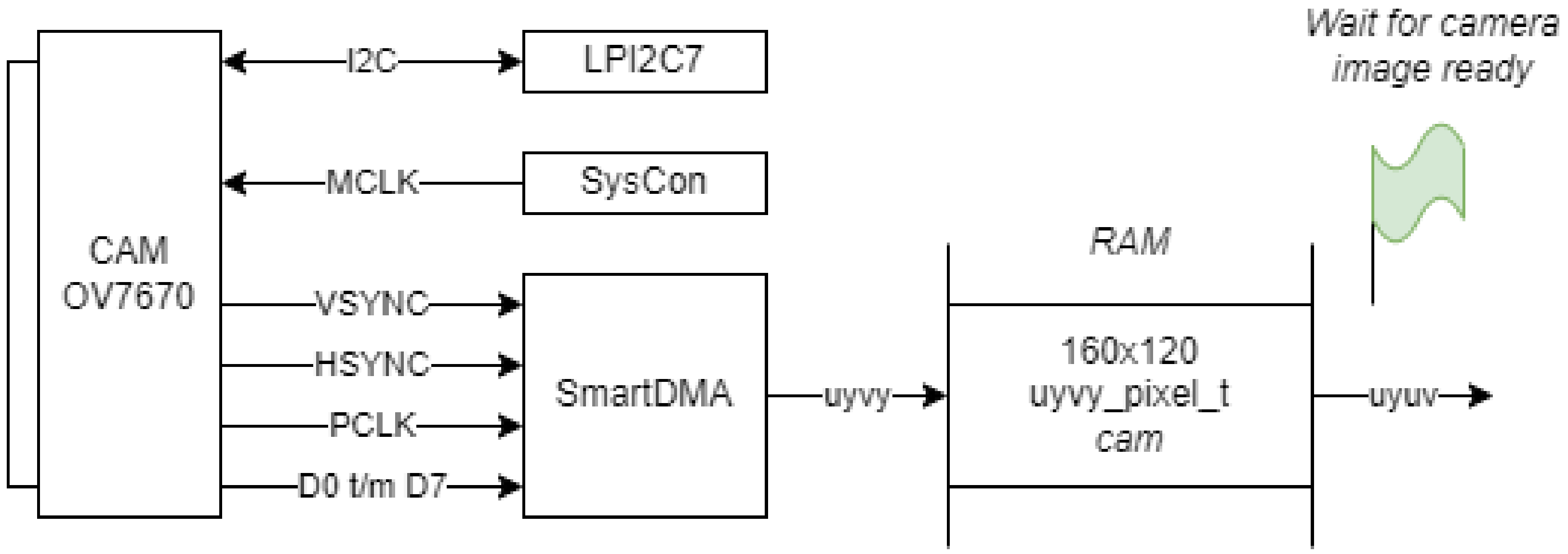
How to interface the OV7670 to the microcontroller?



HAN_UNIVERSITY OF APPLIED SCIENCES

SmartDMA

How to interface the OV7670 to the microcontroller?



The flag is set by means of a callback function

HAN_UNIVERSITY OF APPLIED SCIENCES

SmartDMA

- SmartDMA is a core that implements a reduced instruction set
- It works in a similar way to the ARM core. Being the controller of AHB matrix, SmartDMA can access
  - Registers in modules
  - The GPIO peripheral control and data registers
- To reduce complexity, NXP provides SmartDMA example code for the OV7670 camera module. This code comes in the form of an array called **s_smartdmaCameraFirmware** (see *fsl_smartdma_mcxn.c*).
- This firmware contains absolute jump instructions, so it must be located at a specific address SMARTDMA_CAMERA_MEM_ADDR = 0x04000000U
- This address is the start of the so-called **SRAMX** section
- Code instructions can be fetched from SRAM and this is even faster when executing code from flash!

SmartDMA

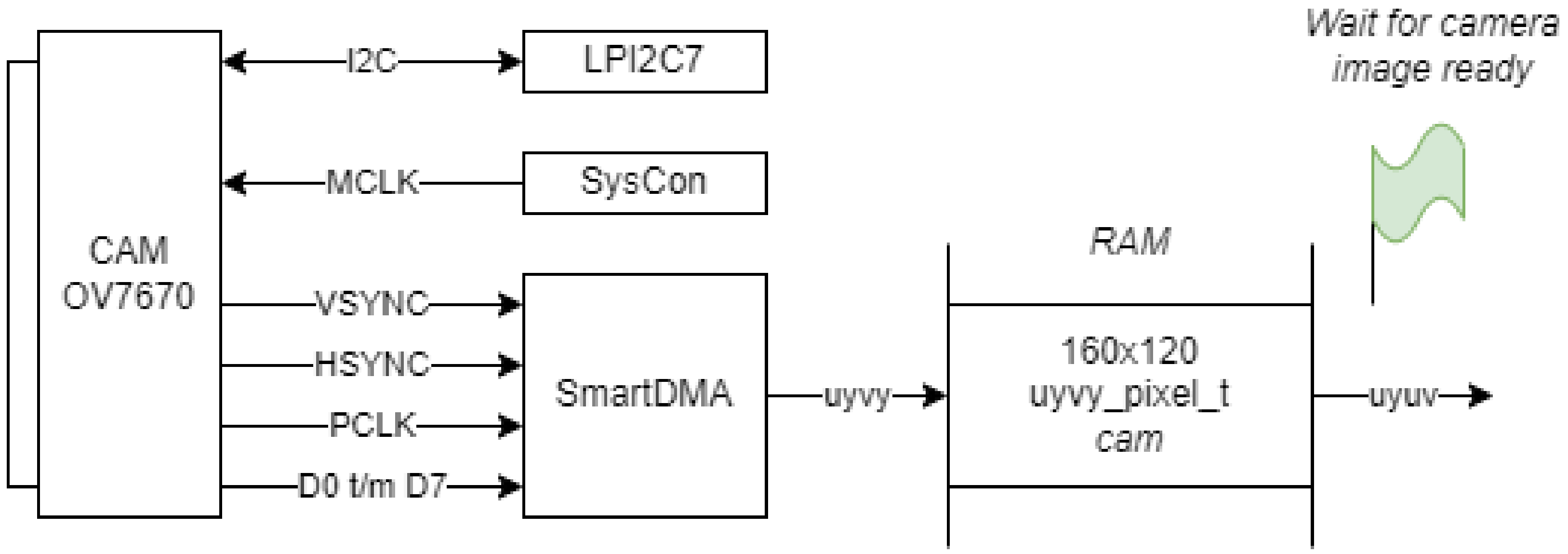
```c
// SmartDMA firmware is copied from FLASH to SRAMX
SMARTDMA_Init(SMARTDMA_CAMERA_MEM_ADDR, s_smartdmaCameraFirmware,
    SMARTDMA_CAMERA_FIRMWARE_SIZE);

// Set the callback function. This function will be called when an entire
// frame from the camera is available.
SMARTDMA_InstallCallback(SmartDMA_camera_callback, NULL);

// Enable SmartDMA interrupts
NVIC_SetPriority(SMARTDMA_IRQn, 0);
NVIC_EnableIRQ(SMARTDMA_IRQn);

// SmartDMA core needs its own stack. The file fsl_smartdma_mcxn.h
// describes that is shall be at least 64 bytes.
smartdma_param.cameraParam.smartdma_stack = (uint32_t *)smartdma_stack;
// Configure pointer for storing camera data
smartdma_param.cameraParam.p_buffer = (uint32_t *)(cam->data);
// Boot the SMARTDMA to run program.
SMARTDMA_Boot(kSMARTDMA_FlexIO_CameraWholeFrame, &smartdma_param, 0x2);
```

SmartDMA

```c
static volatile uint32_t smartdma_camera_image_complete = 0;

static void SmartDMA_camera_callback(void *param)
{
    smartdma_camera_image_complete = 1;
}

while(1U)
{
    // Wait for camera image ready
    while(smartdma_camera_image_complete == 0)
    {}

    // Clear the flag
    smartdma_camera_image_complete = 0;

    // Etc.
}
```

SmartDMA

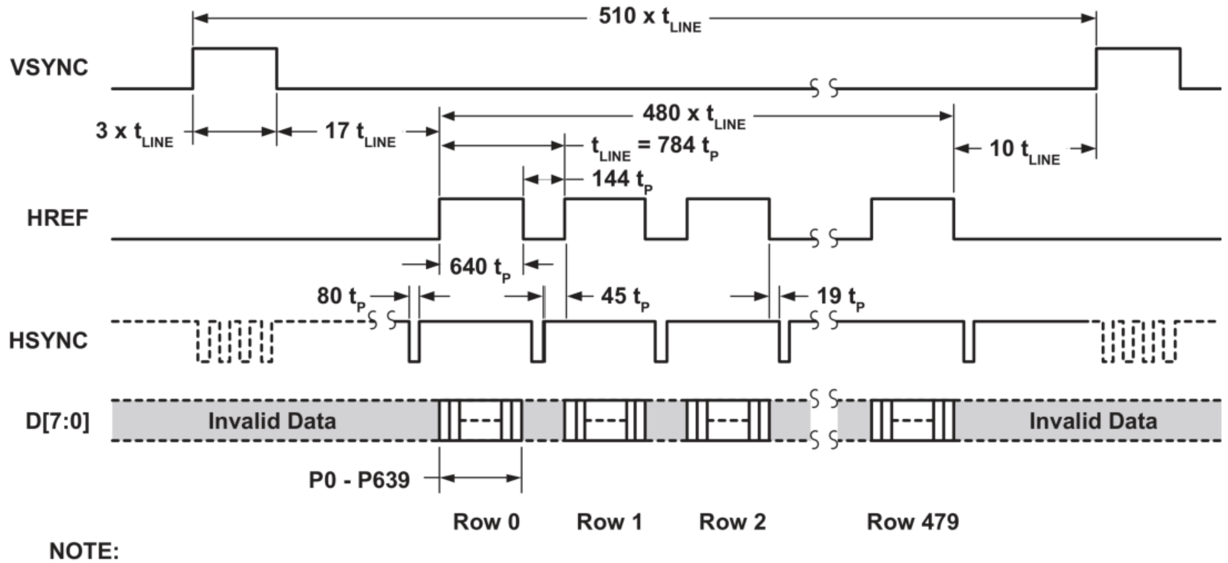
How much time before image data is overwritten?



SmartDMA

How much time before image data is overwritten?

Figure 6 VGA Frame Timing



NOTE:
For Raw data, $t_P = t_{PCLK}$
For YUV/RGB, $t_P = 2 \times t_{PCLK}$

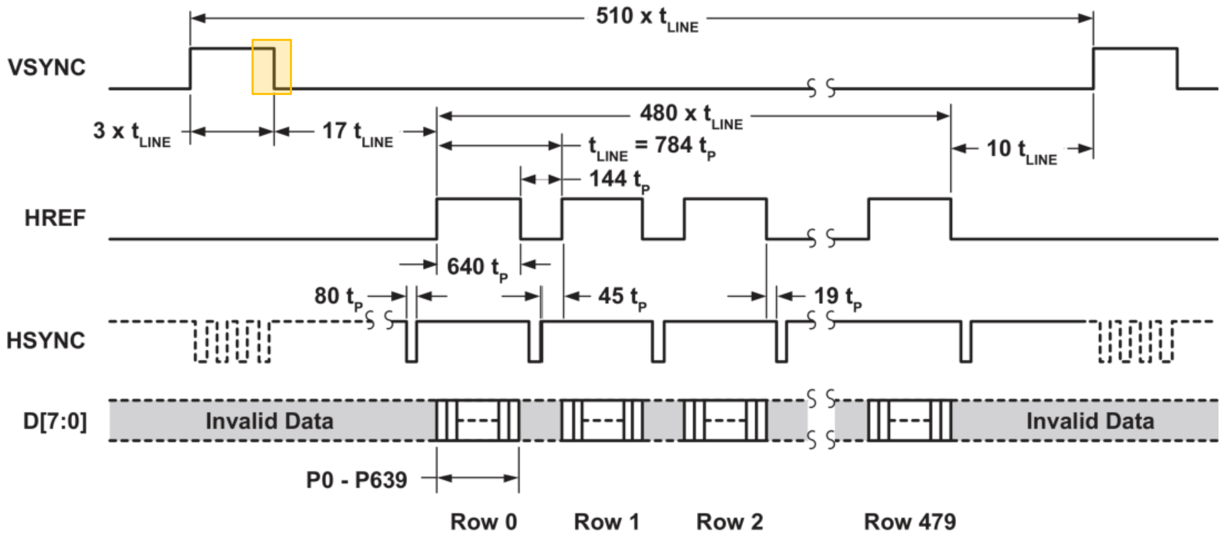
7670CSP_DS_006

SmartDMA

How much time before image data is overwritten?

Let’s assume the worst-case scenario: the DMA controller synchronises on falling edge of VSYNC

Figure 6 VGA Frame Timing



NOTE:
For Raw data, $t_P = t_{PCLK}$
For YUV/RGB, $t_P = 2 \times t_{PCLK}$

7670CSP_DS_006

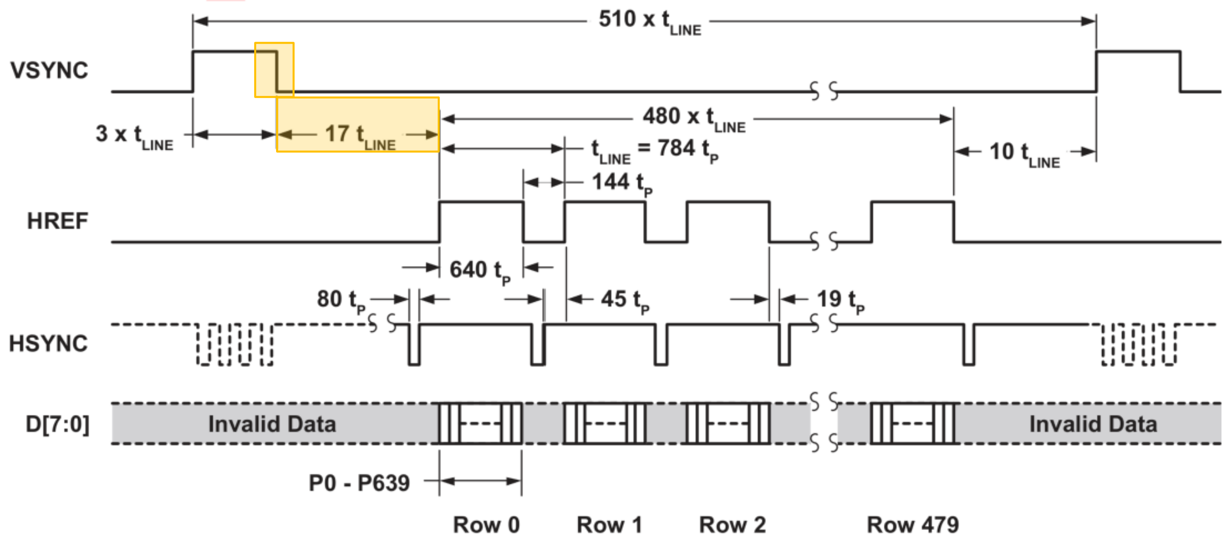
HAN_UNIVERSITY OF APPLIED SCIENCES

SmartDMA

How much time before image data is overwritten?

$17 \times t_{LINE}$ before data is clocked in

Figure 6 VGA Frame Timing



NOTE:
For Raw data, $t_P = t_{PCLK}$
For YUV/RGB, $t_P = 2 \times t_{PCLK}$

7670CSP_DS_006

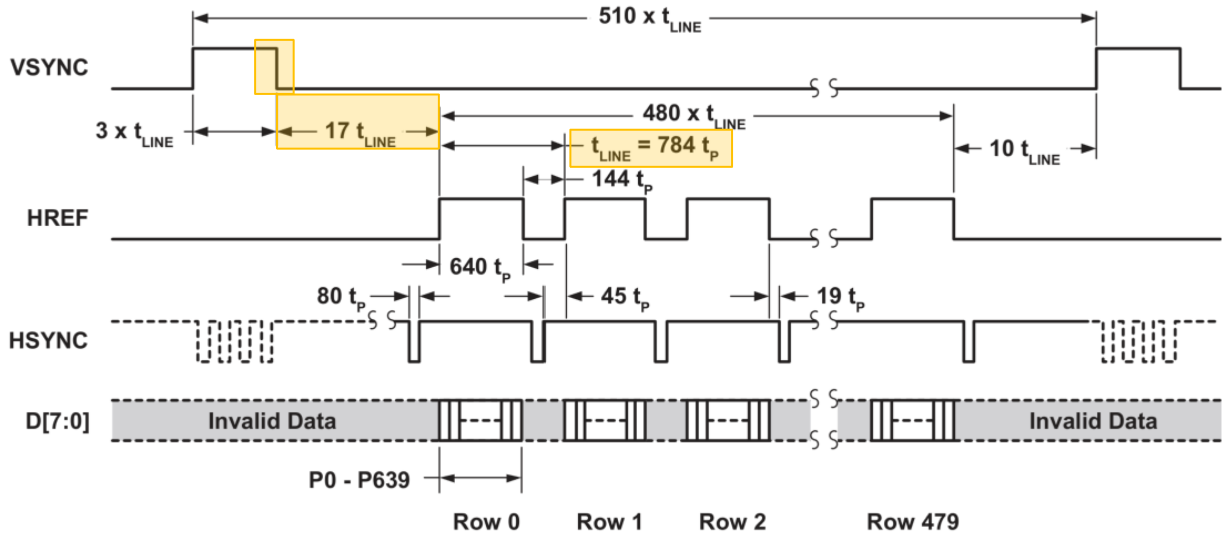
HAN_UNIVERSITY OF APPLIED SCIENCES

SmartDMA

How much time before image data is overwritten?

$$t_{LINE} = 784 \times t_P$$

Figure 6 VGA Frame Timing



NOTE:
For Raw data, $t_P = t_{PCLK}$
For YUV/RGB, $t_P = 2 \times t_{PCLK}$

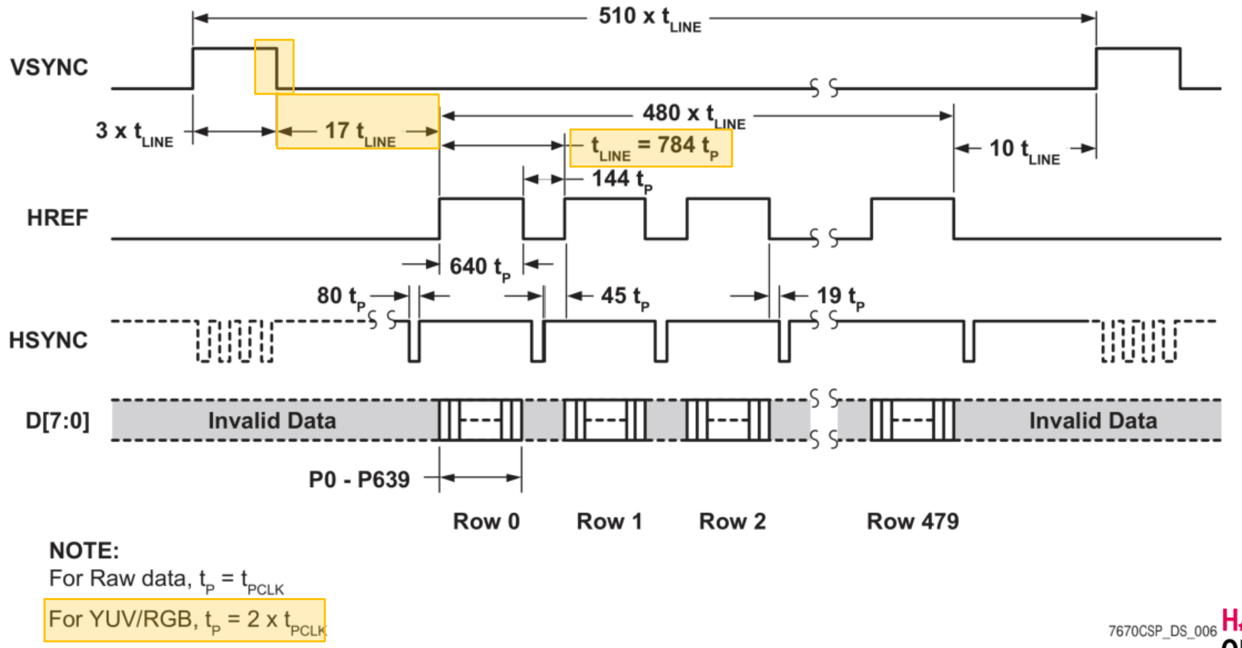
7670CSP_DS_006

SmartDMA

How much time before image data is overwritten?

$$t_{LINE} = 784 \times t_P = 784 \times 2 \times t_{PCLK}$$

Figure 6 VGA Frame Timing



NOTE:
For Raw data, $t_P = t_{PCLK}$
For YUV/RGB, $t_P = 2 \times t_{PCLK}$

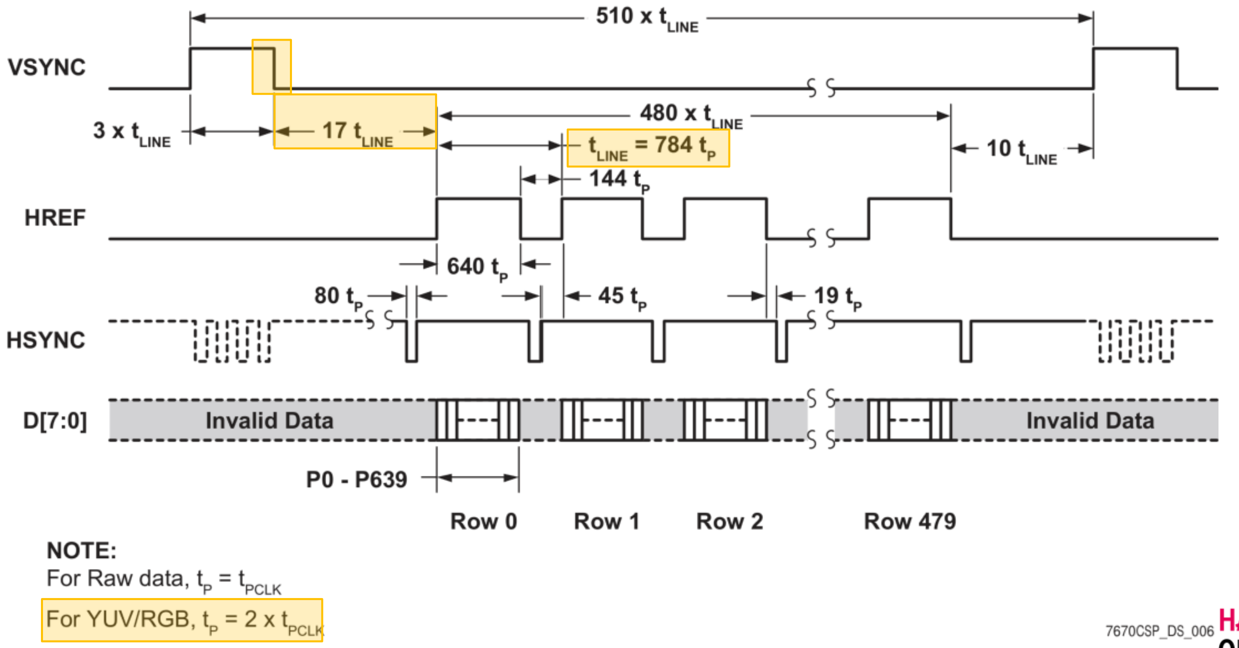
7670CSP_DS_006

SmartDMA

How much time before image data is overwritten?

$$t_{LINE} = 784 \times t_P = 784 \times 2 \times t_{PCLK} = 784 \times 2 \times (1/3MHz) \approx 0.52ms$$

Figure 6 VGA Frame Timing



SmartDMA

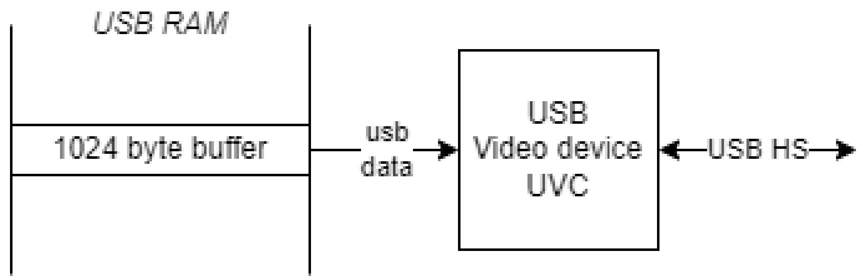
How much time before image data is overwritten?

$$17 \times t_{LINE} = 17 \times 0.52ms = 8.8ms$$

Conclusion: the application must copy the image data within 8.8ms

UVC

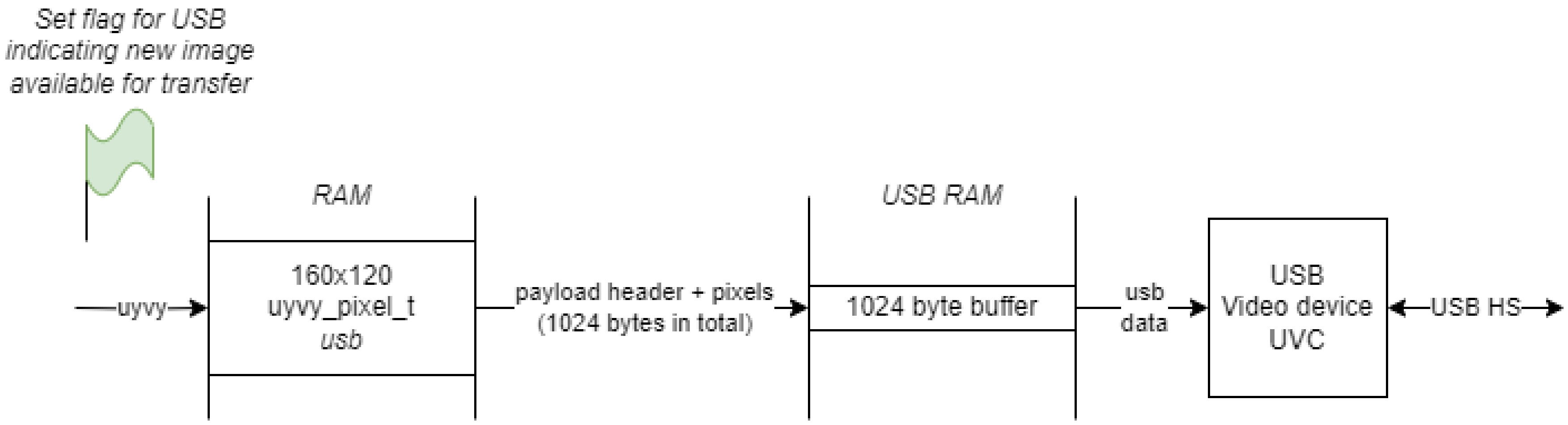
- UVC camera: USB Video Class camera
- Is a video streaming device, which is widely used in webcams, camcorders, still-image camera’s etc.
- MCXN947 features a USB module in High Speed mode
- NXP provides USB video class device drivers



HAN_UNIVERSITY OF APPLIED SCIENCES

UVC

- The function **`USB_DeviceVideoPrepareVideoData()`** is called periodically by the USB driver
- The function checks a flag if a new image is available
- If there is a new image or not all pixel data is transmitted:
  - Copy usb payload header
  - Copy pixel data in remaining space and increment pixel counter

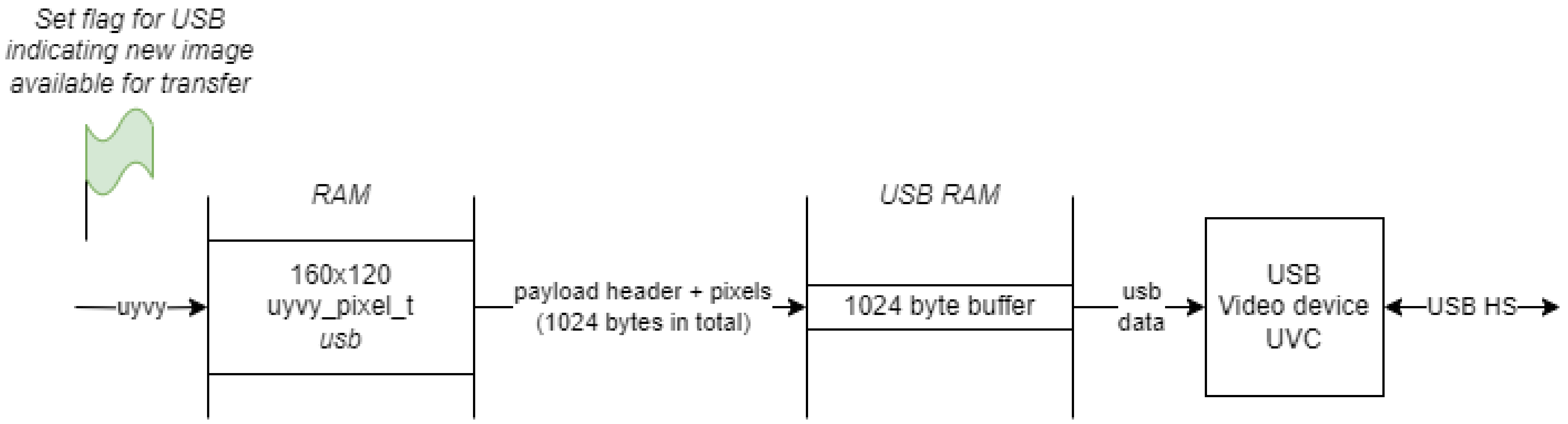


UVC

- To transmit UYVY images, configure the UVC for the following media type (see *usb_device_descriptor.c*):

  GUID = 59565955-0000-0010-8000-00AA00389B71

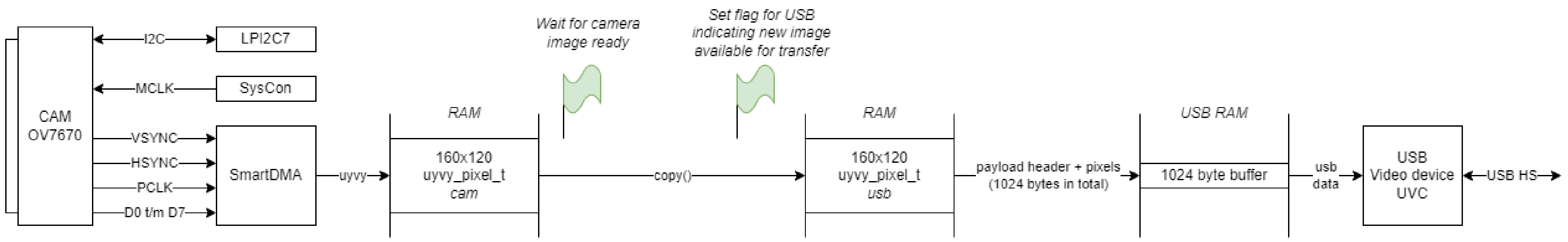
  As defined by Media Foundation and DirectShow Media Types



HAN_UNIVERSITY OF APPLIED SCIENCES

UVC

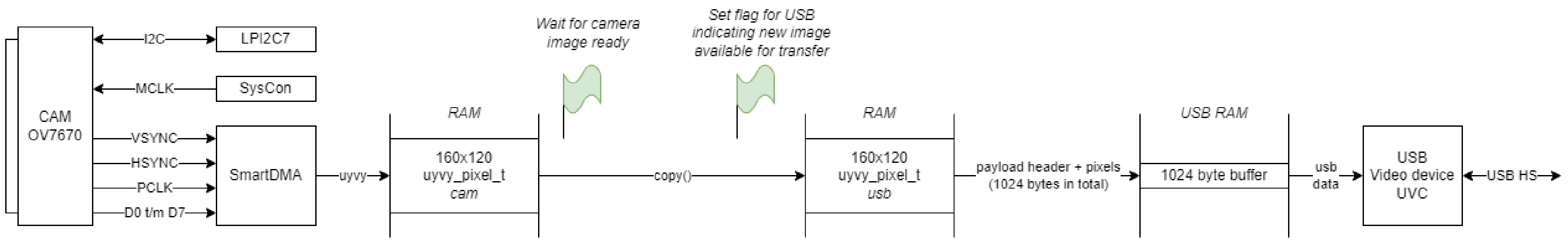
- And, combining the OV7670, SmartDMA and USB modules, we have a working system: the FRDM-MCXN947 is a UVC compliant webcam!



HAN_UNIVERSITY OF APPLIED SCIENCES

UVC

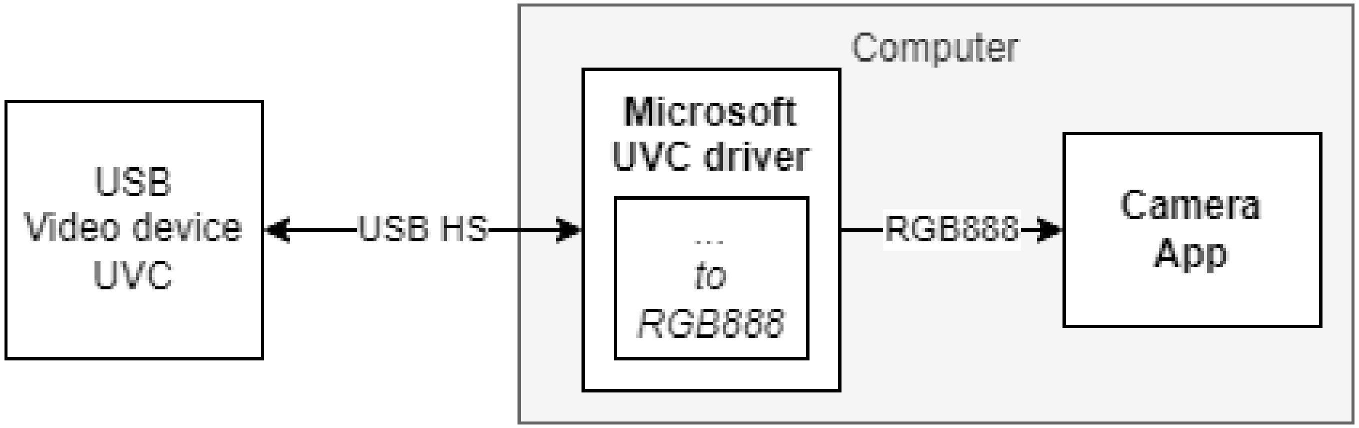
- And, combining the OV7670, SmartDMA and USB modules, we have a working system: the FRDM-MCXN947 is a UVC compliant webcam!



However…

YUV to RGB conversion

- Windows applications, such as the Camera app, use RGB888 images
- More specifically, it uses *Studio video RGB*
- Conversion of the chroma values is specified in the ITU-R BT.601-7 recommendation
- Any input video format is therefore converted to RGB888 by the Microsoft UVC driver



YUV to RGB conversion

- This conversion is implemented by the Windows UVC driver with the following formulas and coefficients:

```
C = Y - 16
D = U - 128
E = V – 128

R = clip(round( 1.164383 * C                         + 1.596027 * E  ))
G = clip(round( 1.164383 * C - (0.391762 * D) - (0.812968 * E) ))
B = clip(round( 1.164383 * C +  2.017232 * D                   ))
```

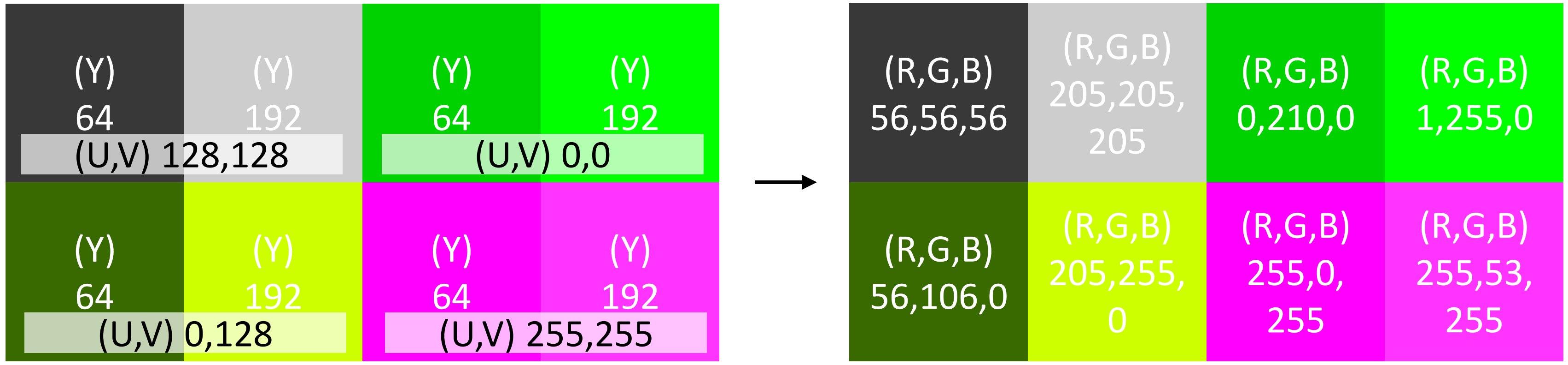
YUV to RGB conversion

- And Microsoft believes “these formulas can be reasonably approximated” by

```
R = clip(( 298 * C           + 409 * E + 128) >> 8)
G = clip(( 298 * C - 100 * D - 208 * E + 128) >> 8)
B = clip(( 298 * C + 516 * D           + 128) >> 8)
```

YUV to RGB conversion

Example Microsoft conversion results



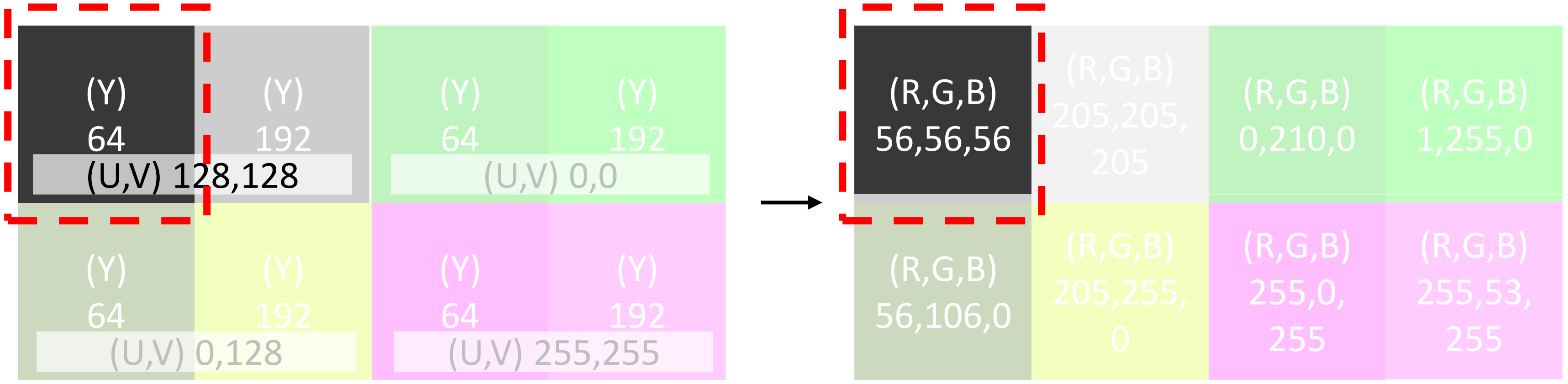
(Y) 64	(Y) 192	(Y) 64	(Y) 192
(U,V) 128,128		(U,V) 0,0	
(Y) 64	(Y) 192	(Y) 64	(Y) 192
(U,V) 0,128		(U,V) 255,255	

$\longrightarrow$

(R,G,B) 56,56,56	(R,G,B) 205,205, 205	(R,G,B) 0,210,0	(R,G,B) 1,255,0
(R,G,B) 56,106,0	(R,G,B) 205,255, 0	(R,G,B) 255,0, 255	(R,G,B) 255,53, 255

YUV to RGB conversion

Weird result: a grayscale value of **64** is converted to **56**!?



Consequently, converting this image in OpenCV to a CV_8UC1, the pixel reads a value of **56**

YUV to RGB conversion

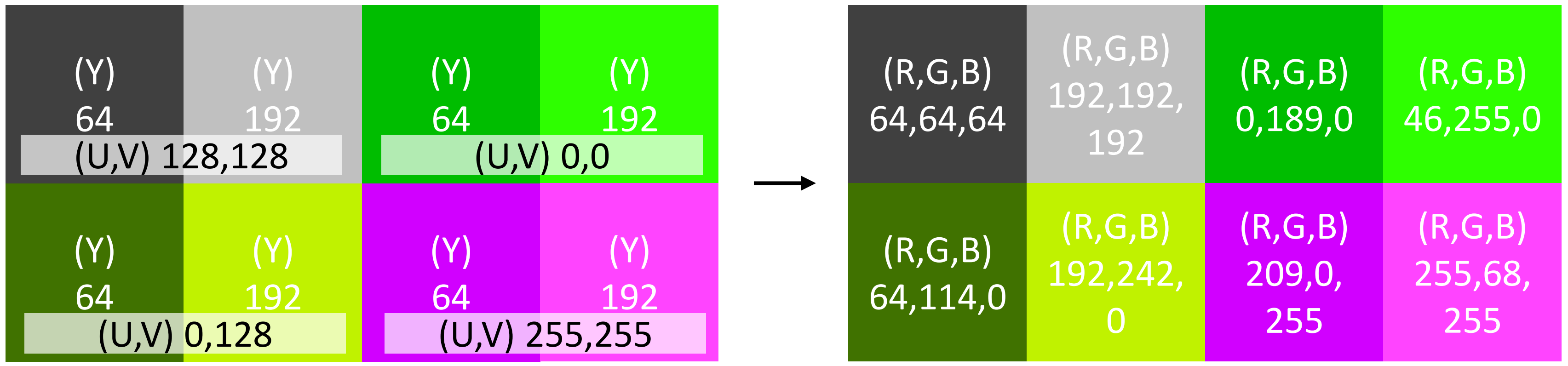
Are there other conversion formula’s and/or coefficients?

Yes, Of course! For example, as implemented by NVIDIA

```
R = clip( Y + 1.140 * (V - 128) )
G = clip( Y - 0.394 * (U - 128) - (0.581 * (V - 128)) )
B = clip( Y + 2.032 * (U - 128) )
```

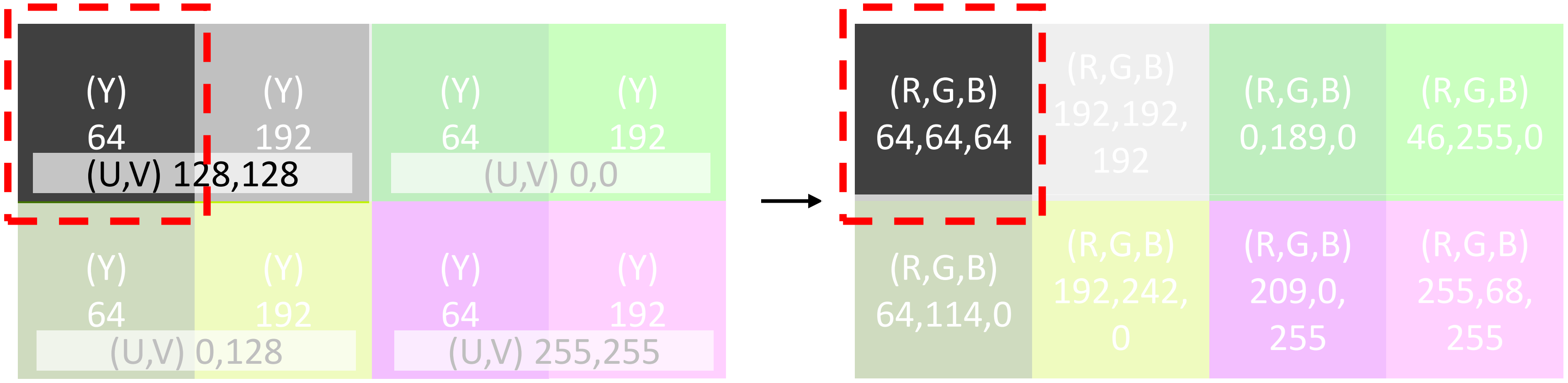
YUV to RGB conversion

Example NVIDIA conversion results



YUV to RGB conversion

Example NVIDIA conversion results



YUV to RGB conversion

Comparing NVIDIA and Microsoft conversion results

(Y) 64	(Y) 192	(Y) 64	(Y) 192
(U,V) 128,128		(U,V) 0,0	
(Y) 64	(Y) 192	(Y) 64	(Y) 192
(U,V) 0,128		(U,V) 255,255	

(R,G,B) 64,64,64	(R,G,B) 192,192, 192	(R,G,B) 0,189,0	(R,G,B) 46,255,0
(R,G,B) 64,114,0	(R,G,B) 192,242, 0	(R,G,B) 209,0, 255	(R,G,B) 255,68, 255

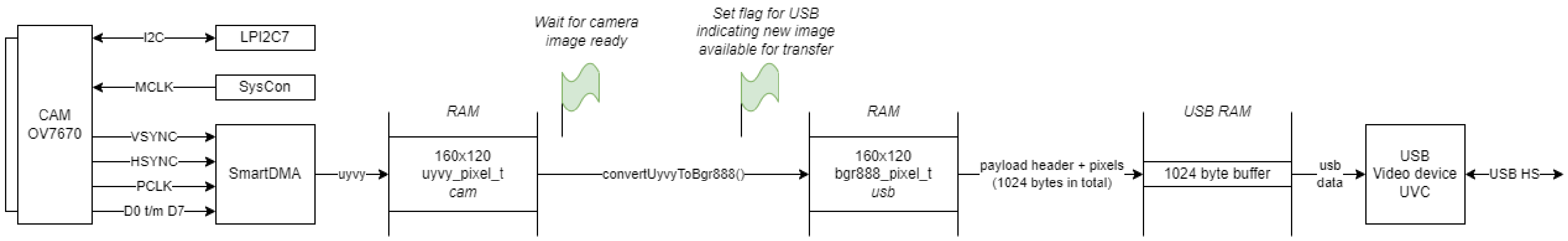
(R,G,B) 56,56,56	(R,G,B) 205,205, 205	(R,G,B) 0,210,0	(R,G,B) 1,255,0
(R,G,B) 56,106,0	(R,G,B) 205,255, 0	(R,G,B) 255,0, 255	(R,G,B) 255,53, 255

See the function

`convertUyvyToBgr888()`

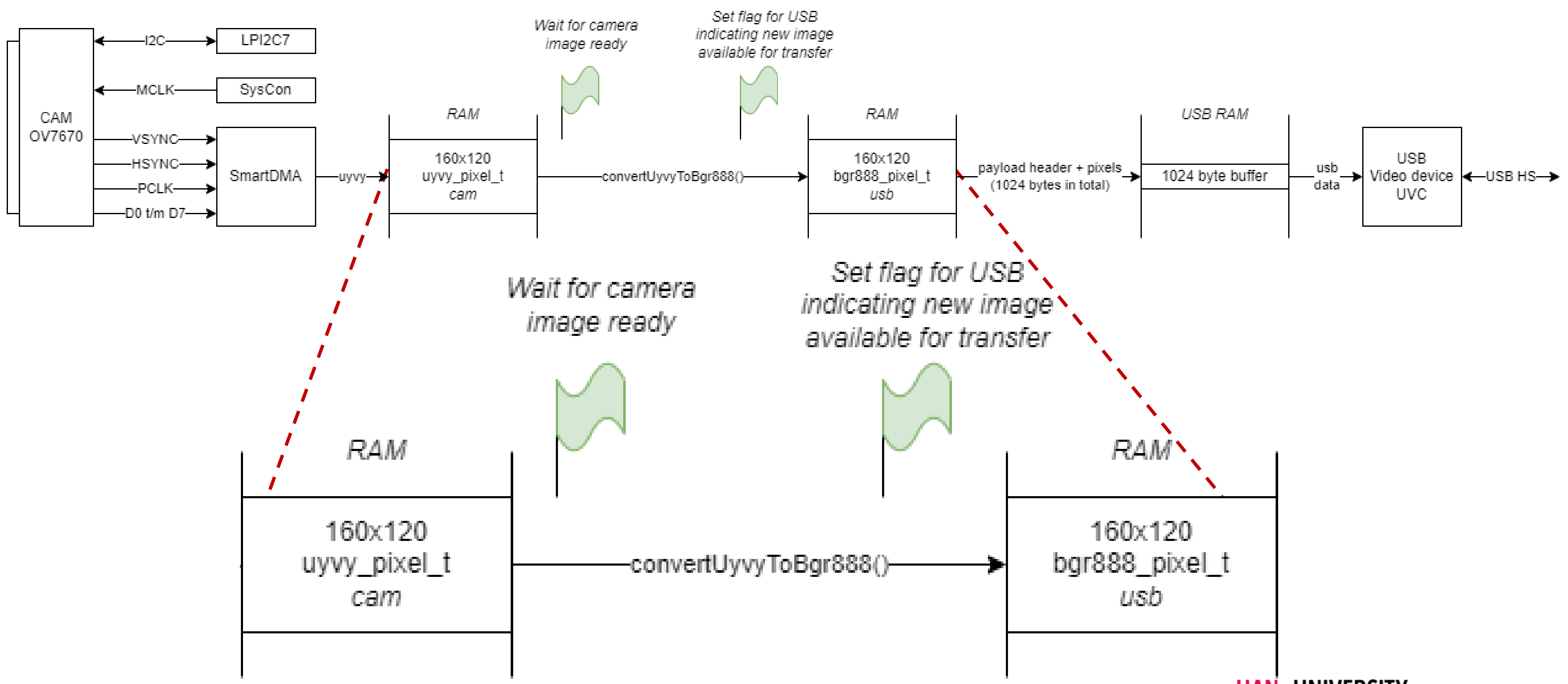
RGB888

Conclusion: If we cannot rely on the RGB conversion on the PC, we must do the conversion ourselves on the microcontroller!



RGB888

Conclusion: If we cannot rely on the RGB conversion on the PC, we must do the conversion ourselves on the microcontroller!

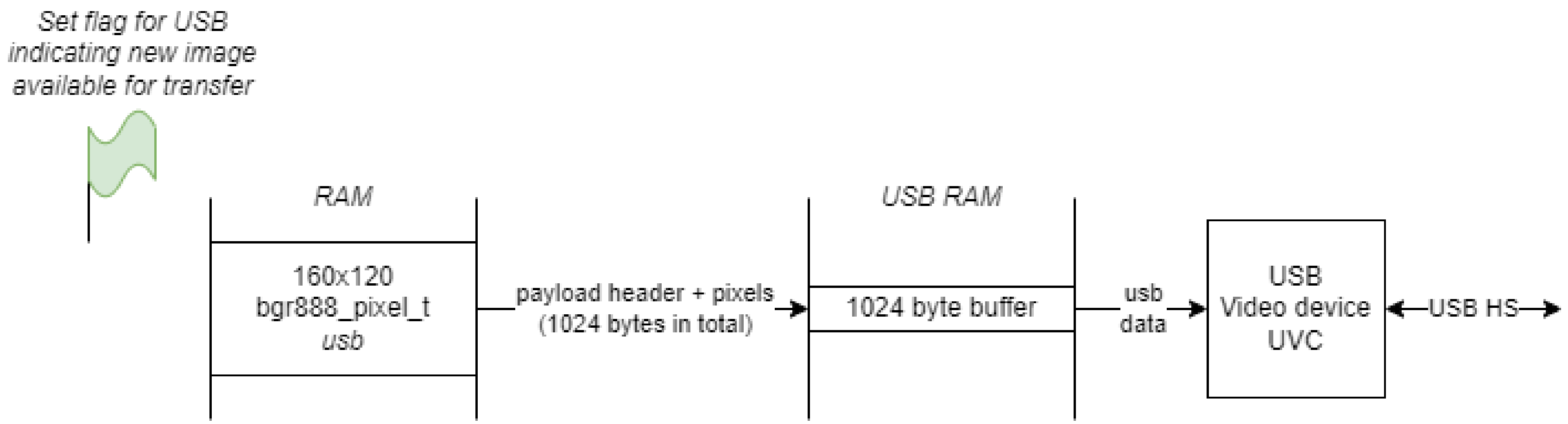


RGB888

- To transmit RGB888 (a.k.a. RGB24) images, configure the UVC for the following media type (see *usb_device_descriptor.c*):

  GUID = E436EB7D-524F-11CE-9F53-0020AF0BA770

  As defined by [Media Foundation and DirectShow Media Types]



RGB888 or BGR888?

The Media Foundation and DirectShow Media Types defines that all formats are listed from **left** to **right**, where left is **MSB** and right is **LSB**

So for the RGB888 (RGB 24) format:

- R is stored in MSB
- B is stored in LSB

```
typedef struct
{
    uint8_t b; ///< blue   LSB
    uint8_t g; ///< green
    uint8_t r; ///< red    MSB

}how_to_call_this_struct_t; // ???
```

HAN_UNIVERSITY OF APPLIED SCIENCES

RGB888 or BGR888?

I decided to use the BGR888 naming, for two reasons:

1. Is consistent with other EVDK naming conventions, such as uyvy_pixel_t (also lsb first)
2. OpenCV VideoCapture.read() function returns a BGR888 image

```
/// \brief Type definition of an BGR888 pixel
///
/// 3*8=24 bits per pixel
typedef struct
{
    uint8_t b; ///< blue
    uint8_t g; ///< green
    uint8_t r; ///< red

}bgr888_pixel_t;
```

YUV to BGR888 conversion

Code example

```
while(1U)
{
    // Wait for camera image complete
    while(smartdma_camera_image_complete == 0)
    {}

    smartdma_camera_image_complete = 0;

    // Copy and convert image to BGR888 USB image buffer
    convertUyvyToBgr888(cam, usb);

    // Set flag for USB indicating new image available for transfer
    image_available_for_usb = 1;
}
```

HAN_UNIVERSITY OF APPLIED SCIENCES

YUV to BGR888 conversion

Code example

```c
while(1U)
{
    // Wait for camera image complete
    while(smartdma_camera_image_complete == 0)
    {}

    smartdma_camera_image_complete = 0;

    // Copy and convert image to BGR888 USB image buffer
    convertUyvyToBgr888(cam, usb);

    // Set flag for USB indicating new image available for transfer
    image_available_for_usb = 1;
}
```

The repeat rate of this loop is determined by the refresh rate of the OV7670 camera.

Four frame rates are supported:

- **14 fps – 71ms between frames**
- **15 fps – 66ms between frames**
- **25 fps – 40ms between frames**
- **30 fps – 33ms between frames**

YUV to BGR888 conversion

Code example

```
while(1U)
{
    // Wait for camera image complete
    while(smartdma_camera_image_complete == 0)
    {}

    smartdma_camera_image_complete = 0;

    // Copy and convert image to BGR888 USB image buffer
    convertUyvyToBgr888(cam, usb);

    // Set flag for USB indicating new image available for transfer
    image_available_for_usb = 1;
}
```

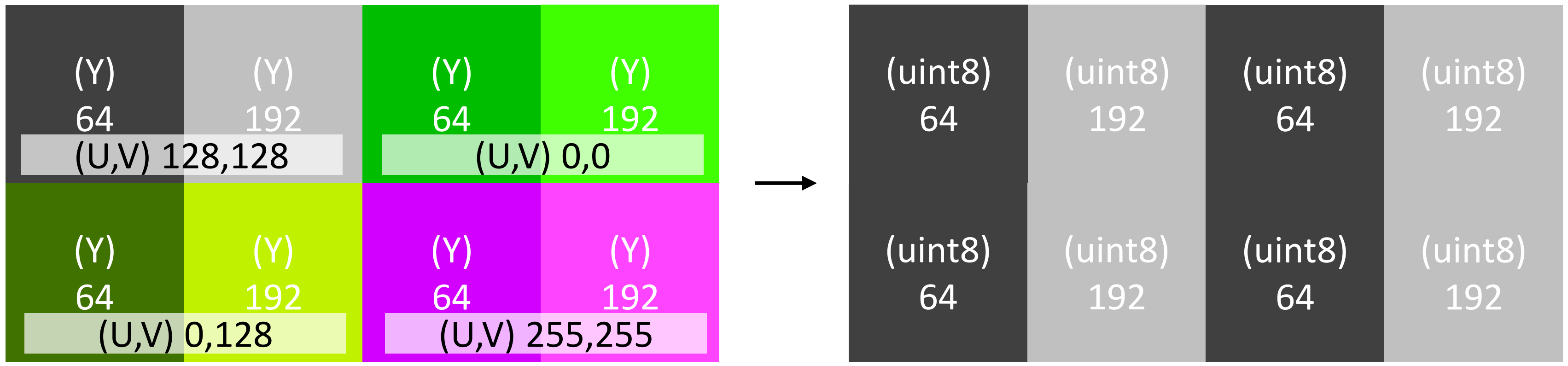
5ms ~ 6ms

HAN_UNIVERSITY OF APPLIED SCIENCES

What other conversions are available?

YUV to UINT8 conversion

YUV to uint8 conversion is achieved by discarding the U-V information:

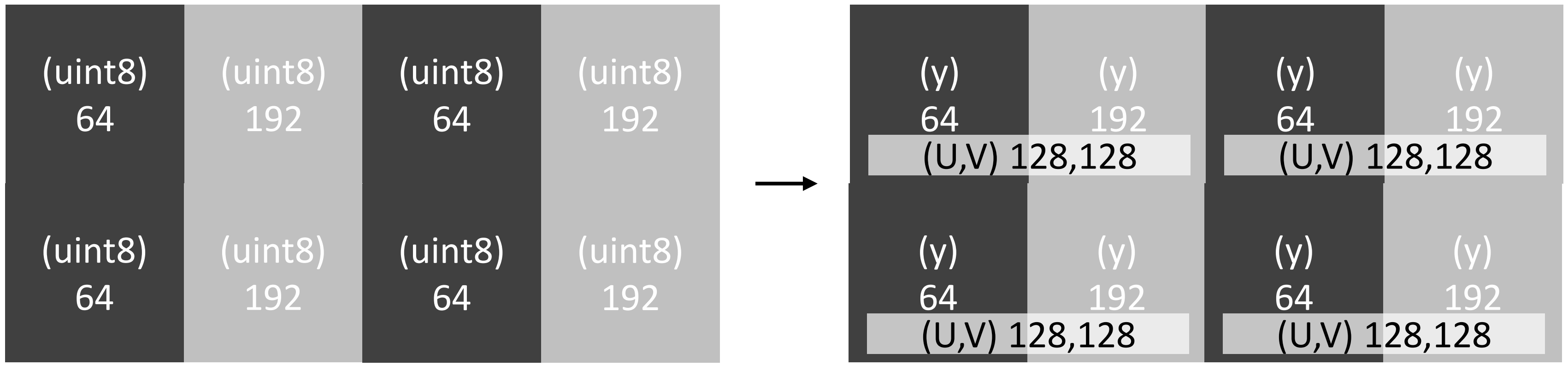


See the function **`convertUyvyToUint8()`**

1ms ~ 2ms

UINT8 to YUV conversion

Or the other way around, by adding the U-V values and set these to 0 (128):



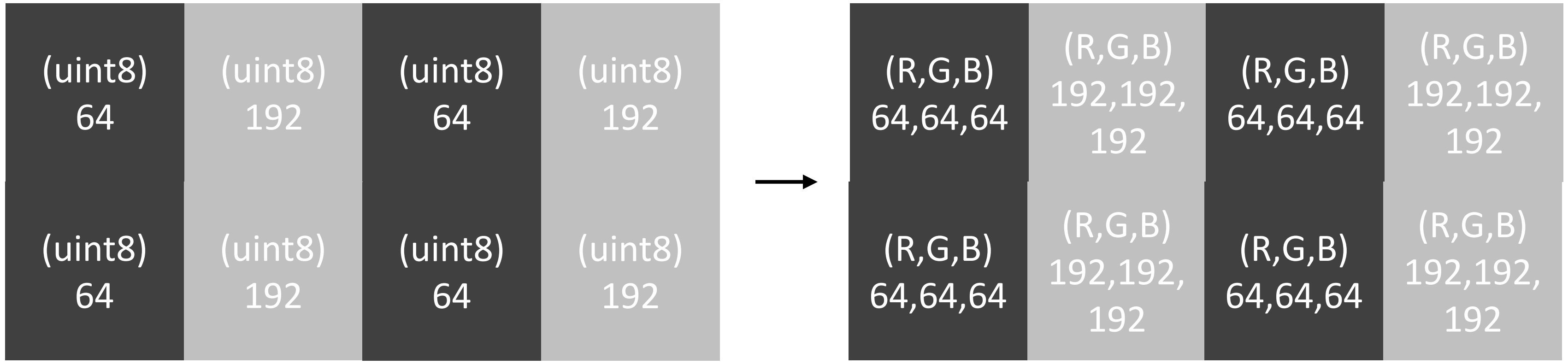
See the function **`convertUint8ToUyvy()`**

1ms ~ 2ms

HAN_UNIVERSITY OF APPLIED SCIENCES

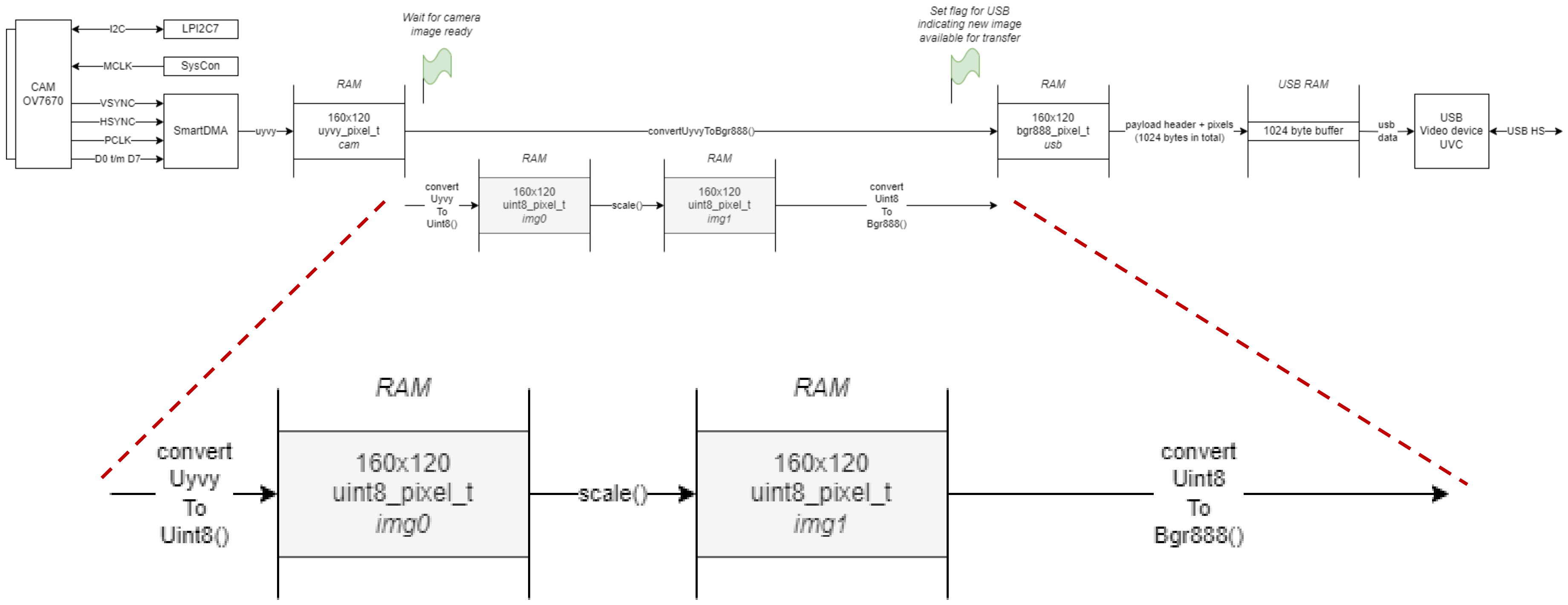
UINT8 to RGB conversion

Or to RGB888, by assigning the same value to all channels



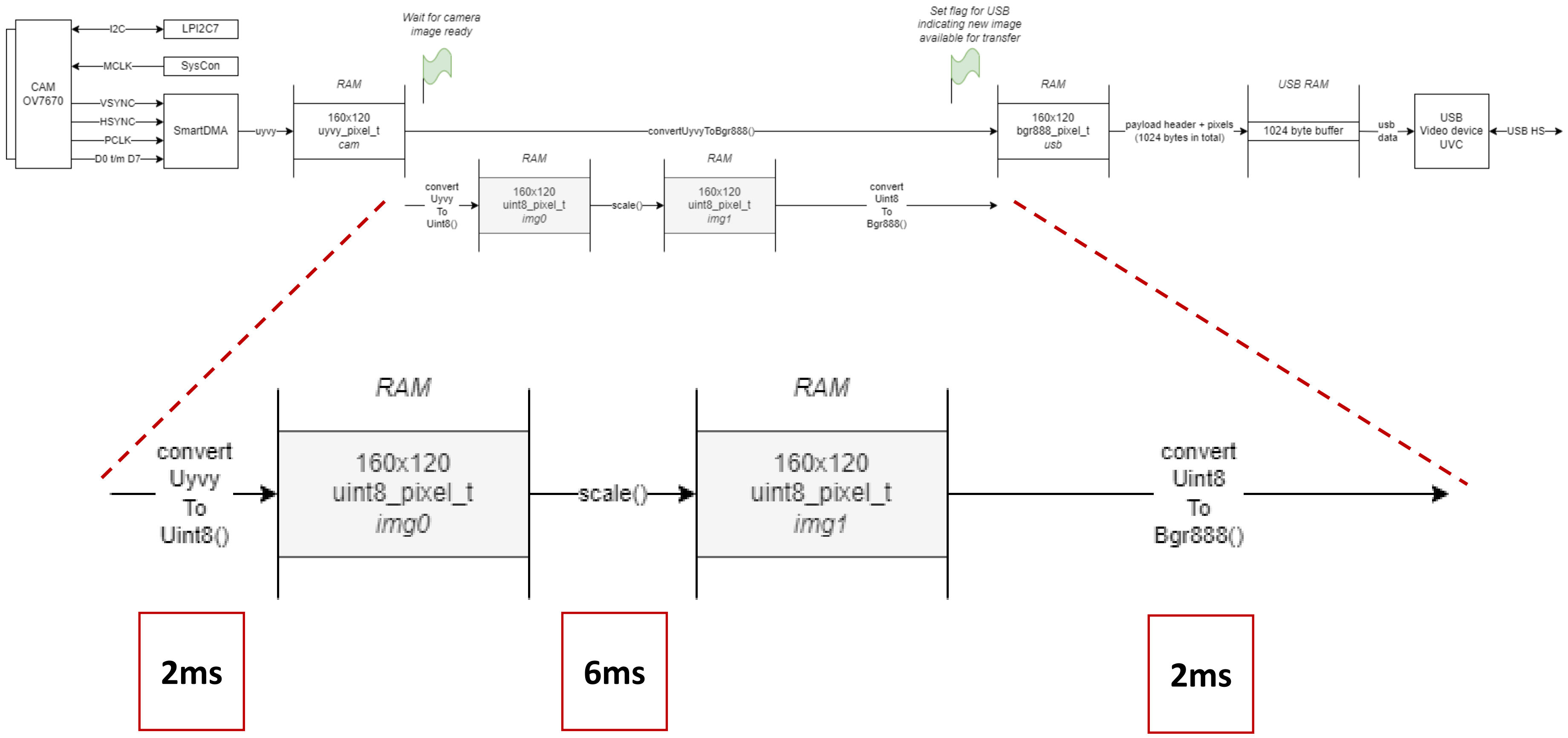
See the function `convertUint8ToBgr888()` **1ms ~ 2ms**

Image processing pipeline



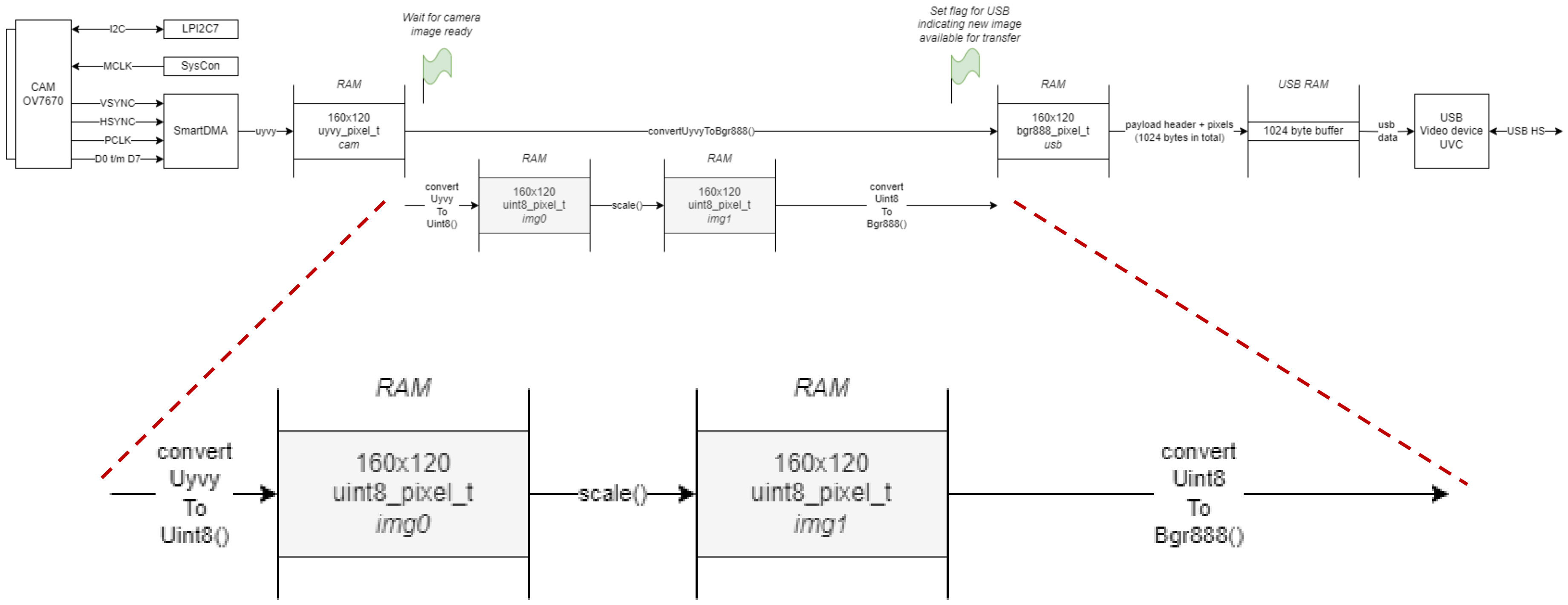
HAN_UNIVERSITY OF APPLIED SCIENCES

Image processing pipeline


CAM OV7670
I2C
LPI2C7
MCLK
SysCon
VSYNC
HSYNC
PCLK
D0 t/m D7
SmartDMA
uyvy
RAM
160x120 uyvy_pixel_t cam
Wait for camera image ready
convertUyvyToBgr888()
Set flag for USB indicating new image available for transfer
RAM
160x120 bgr888_pixel_t usb
payload header + pixels (1024 bytes in total)
USB RAM
1024 byte buffer
usb data
USB Video device UVC
USB HS
convert Uyvy To Uint8()
RAM
160x120 uint8_pixel_t img0
scale()
RAM
160x120 uint8_pixel_t img1
convert Uint8 To Bgr888()
2ms
6ms
2ms

HAN_UNIVERSITY OF APPLIED SCIENCES

Image processing pipeline



See the example

`exampleWebcamUint8()`

EVD1 – Assignment

Study guide

Week 1

7 Image fundamentals – convertUyvyToUint8()

HAN_UNIVERSITY
OF APPLIED SCIENCES

References

- NXP Semiconductors. (2017). *AN12103 - Developing a simple UVC device based on i.MX RT1050.* Retrieved May 7, 2024, from https://www.nxp.com/docs/en/application-note/AN12103.pdf

- NXP Semiconductors. (2024). *AN14191 - How to Use SmartDMA to Implement Camera Interface in MCXN MCU.* Retrieved May 7, 2024, from https://www.nxp.com/docs/en/application-note/AN14191.pdf

- Wikipedia contributors. (2024, April 22). Y′UV. In *Wikipedia, The Free Encyclopedia*. Retrieved 18:18, May 14, 2024, from https://en.wikipedia.org/w/index.php?title=Y%E2%80%B2UV&oldid=1220238267

References

- ITU-R. (06/2015). *Recommendation ITU-R BT.601-7.* Retrieved September 18, 2024, from *https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf*

- NVIDIA. (2004). YUVToRGB – Color Model Conversion. In *NVIDIA 2D Image And Signal Performance Primitives (NPP) Version 10.2.*.**. Retrieved September 16, 2024, from https://docs.nvidia.com/cuda/archive/10.1/npp/group__yuvtorgb.html

- Microsoft Media Foundation. (07/2021). *Recommended 8-Bit YUV Formats for Video Rendering.* Retrieved September 18, 2024, from https://learn.microsoft.com/en-us/windows/win32/medfound/recommended-8-bit-yuv-formats-for-video-rendering

References

- Wikipedia contributors. (2024, September 18). Chroma subsampling. In *Wikipedia, The Free Encyclopedia*. Retrieved 09:30, September 23, 2024, from https://en.wikipedia.org/wiki/Chroma_subsampling