Getting video stream from USB web-camera on Arduino Due - Part 1: Getting Started

Feb 14, 2015 11 min read words

C Dev Intermediate hardware video USB Arduino UVC ARM

by Sergiy Bogdancev
Contributor

Introduction

The purpose of this project is to show how to get video stream from USB camera. However, one can also learn how to work with various parts of ARM Cortext-M3 SAM3X device, understand something about USB 2.0 and USB Video Class (UVC). I will explain what each piece of code does and what the correct sequences for various tasks are.

As you read those articles please note that this is my first ever embedded C project, my first ever Arduino Due project, my first ever USB and UVC project and my language of origin is not English.

Having already completed the project (it took me about half a year) I can now tell that Arduino Due is not enough to handle video, I suspected that at the beginning but I said myself I should

way I think you won't be able to get colour video as even uncompressed video stream requires lots of recalculations.

Let's start from listing the project's parts.

Project hardware

Besides personal computer following hardware is used:

- Arduino Due;
- 320x240 TFT monitor shield, take a note that this particular monitor is very dull and its surface like a mirror - a very bad monitor as for me or perhaps I don't know how to initialize it correctly. It would be very nice if someone could give more input about such TFT monitors;

Project software

I used free tools or free versions only:

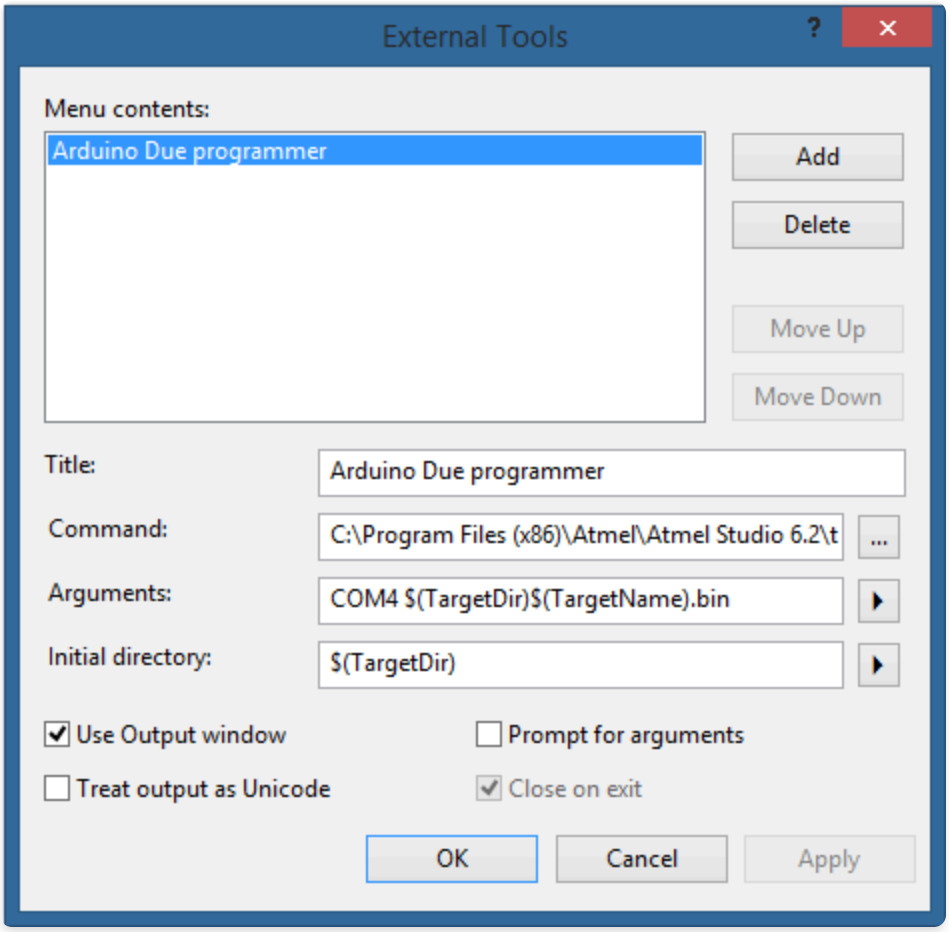
- Atmel Studio 6.x to write my code. It is a free package from Atmel. Looks like Visual Studios from Microsoft. Will require to fill a form before downloading;
- Docklight a RS-232 monitor program to read debug messages from Arduino Due. I use free version that can't do some things which are not important for me in this project. Instead of Docklight one can use any other similar program.
- Arduino software. We will not use Arduino software, it just installs USB driver for communication with the board and has a tool called bossac.exe. This tool is very important as the board is programmed with its help. I will not discuss details how to install it in Atmel Studio because people already described it nicely for example in this post.

Documents

I will refer to following documents in my text:

- Definitive guide to ARM Cortex M3 2nd edition. This one gives you full understanding about Cortex-M3 and some differences to other versions. Not all sections of this big book is necessary to read. It can be legally bought or free downloaded from torrent trackers such as Pirate Bay.
- SAM3X full specification. SAM3X8E is the processor on Arduino Due board. **[1]**
- USB 2.0 specification. There is file usb_20.pdf inside the archive – very important document. **[2]**
- USB Video Class 1.0 specification. I could not find it on their web-site because there are newer versions of this specification (1.1 and 1.5) available but it appeared that the camera I bought adheres to version 1.0 thus I downloaded the document from some Linux forum. **[3]**
- USB Video Payload Uncompressed 1.0 specification. This document describes transport stream that is used by my camera. Your camera can have different transport stream and there is a document for each such stream. I will show later in the text how to find out your camera transport stream. **[4]**

Getting started



Installation

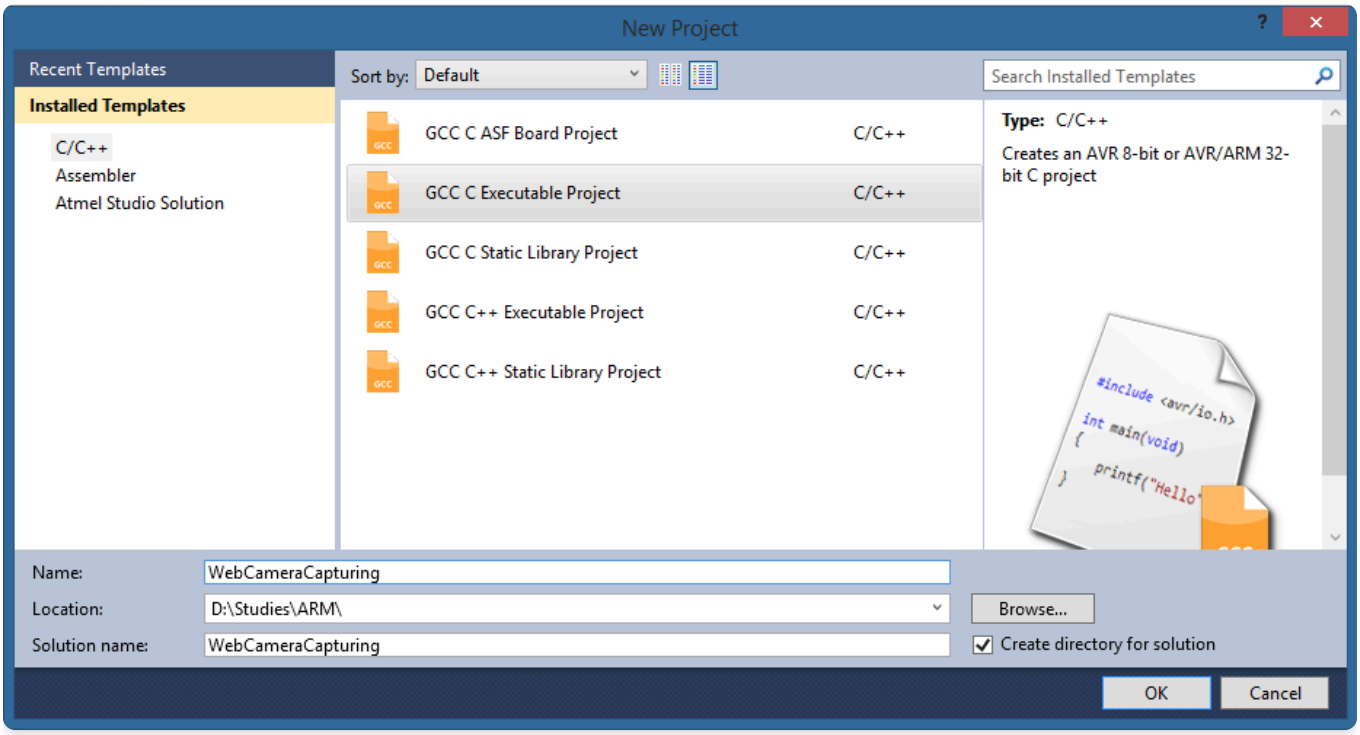
Please install Atmel studio 6.x, Docklight, Arduino software. Connect Arduino Due to computer USB port, wait until OS installs USB drivers (provided in Arduino software folder), in driver's properties check which COM port number it uses for Arduino Due and remember it. Then read carefully above mentioned post about how to install external tool for Arduino Due programming. At the end it should look like mine on the picture. Note that my COM port number is 4, yours can be different.

To use it, simply compile the program with no errors, then hover your mouse over Tools menu and select Arduino Due programmer. Studio will upload your program to Arduino Due using bossac.exe utility.

Please take a note that later we'll use Docklight and because COM port cannot be shared it must be closed in Docklight before programming. *Also every time you open COM port in Doclight restarts Arduino Due which is very useful feature.*

simulator for ARM processors.

Creating new project



Open Atmel Studio, select menu File -> New -> Project, select GCC C Executable Project, call it as you like. I'll refer to file with main function as **main file** further in the text.

In the next window select the target device. Arduino Due has ATSAM3X8E microcontroller, find it in the list, select and click OK.

During this several click process Atmel Studio did some job for us. Let's see.

In /cmsis/src forlder thare are 2 files: startup_sam3xa.c and system_sam3xa.c.

The code in the first file is executed even before our code: it defines vector table, initializes global variables to zero, C library, etc and there is a branch to the main function (that is where our program starts). I think this startup code is considered as advanced topic which I don't know much myself and I will not use it other than looking the correct name for an interrupt (exception). If you want to understand what it does, please read some articles about "bare-metal programming" like this.

In the following sections I will show what needs to be done to initialize Arduino Due, you will see all necessary registers and code sequences. I will also simplify the code created by Atmel Studio and put correct comments

CPP

```
#include "sam.h"

int main(void)
{
    SystemInit();
    while(1);
}
```

"Goto Implementation" of that function. First it initializes flash memory read operations, as Flash runs much slower than the core we need to specify wait states **[1, p.1434]**. From table 45-62 for 84MHz frequency wait state must be 4. It is exactly what is in ***SystemInit().*** Wait states also mean that if a code has conditions and jumps, it will lower run speed as after each such operation the core will have to empty conveyer and wait 4 cycles (in our case) to access next instruction from the memory. Once it accessed the memory, it can fetch 2 or 4 instructions at a time so it does not need to wait again.

CPP

```
EFC0->EEFC_FMR = EEFC_FMR_FWS(4);
EFC1->EEFC_FMR = EEFC_FMR_FWS(4);
```

After startup the microcontroller runs on embedded fast RC oscillator at 4MHz **[1, p.522]**. We need to switch it to run from much more stable 3-20MHz crystal oscillator which has attached 12MHz crystal according to Arduino Due schematic. That is what next piece of code does:

CPP

```
if ( !(PMC->CKGR_MOR & CKGR_MOR_MOSCSEL) )
{
    PMC->CKGR_MOR = CKGR_MOR_KEY_PASSWD | SYS_BOARD_OSCOUNT |
```

```
        }
    }
```

First it checks if 3-20MHz oscillator is not selected, we know that it is not. Then if not selected it enables it. And finally it waits until crystal oscillator comes up and stabilizes. I will drop what we don't need, code now looks like this:

CPP

```
/* Initialize main oscillator */
PMC->CKGR_MOR |= CKGR_MOR_KEY_PASSWD | SYS_BOARD_OSCOUNT | CKGR_MOR_MOSCXTEN;
while ( !(PMC->PMC_SR & PMC_SR_MOSCXTS) );
```

Note that writing to Main oscillator register CKGR_MOR requires password which is value 0x37.

Once 3-20MHz crystal oscillator has been initialized, we switch to it from Embedded RC oscillator and wait until takeover is done:

CODE

```
PMC->CKGR_MOR = CKGR_MOR_KEY_PASSWD | SYS_BOARD_OSCOUNT | CKGR_MOR_MOSCRCEN |
CKGR_MOR_MOSCXTEN | CKGR_MOR_MOSCSEL;
while ( !(PMC->PMC_SR & PMC_SR_MOSCSELS) );
```

Again, I simplify it by using OR not to show other parameters that we already used and to emphasise what exactly this step does:

CPP

```
/* Switch to 3-20MHz Xtal oscillator */
PMC->CKGR_MOR |= CKGR_MOR_KEY_PASSWD | CKGR_MOR_MOSCSEL;
```

oscillator (which is default on startup) or 3-20MHz crystal oscillator which we initialized and selected just above. So we do not need to switch to main clock as it was already used from the beginning and following piece of code can be omitted:

CPP

```
PMC->PMC_MCKR = (PMC->PMC_MCKR & ~(uint32_t)PMC_MCKR_CSS_Msk) |
PMC_MCKR_CSS_MAIN_CLK;
while (!(PMC->PMC_SR & PMC_SR_MCKRDY));
```

What we need is to somehow change clock frequency from 12MHz to 84MHz. For that there is Divider and PLL (phase lock loop) **[1, p.524]**. Next piece of code initializes them:

CPP

```
PMC->CKGR_PLLAR = SYS_BOARD_PLLAR;
while ( !(PMC->PMC_SR & PMC_SR_LOCKA) );
```

Please note that SYS_BOARD_PLLAR is defined on top of this file and supplies values described on page 524 section 27.6.1 of **[1]** and gives 168MHz output. It needs to be divided by 2 to obtain 84MHz, that is what next piece of code does:

CPP

```
/* Switch to main clock */
PMC->PMC_MCKR = (SYS_BOARD_MCKR & ~PMC_MCKR_CSS_Msk) | PMC_MCKR_CSS_MAIN_CLK;
while ( !(PMC->PMC_SR & PMC_SR_MCKRDY) )
{
}
```

As you can see the comment is wrong and code is a bit complicated for this small operation. I will rewrite it as following:

```
/* Setting up prescaler */
PMC->PMC_MCKR = PMC_MCKR_PRES_CLK_2 | PMC_MCKR_CSS_MAIN_CLK;
while ( !(PMC->PMC_SR & PMC_SR_MCKRDY) ) ;
```

It specifies prescaller (division by 2) and preserves Main clock selection. At this point everything is ready to switch to PLLA (84MHz) clock:

CPP

```
/* Switch to PLLA */
PMC->PMC_MCKR = SYS_BOARD_MCKR;
while ( !(PMC->PMC_SR & PMC_SR_MCKRDY) )
{
}
```

Note that I removed SYS_BOARD_MCKR macros and used what it consists of. system_sam3xa.c looks now like this:

CPP

```
#include "sam3xa.h"

#ifdef __cplusplus
extern "C" {
    #endif

    /* Clock settings (84MHz) */
    #define SYS_BOARD_OSCOUNT   (CKGR_MOR_MOSCXTST(0x8))
    #define SYS_BOARD_PLLAR     (CKGR_PLLAR_ONE | CKGR_PLLAR_MULA(0xdUL)
                                 | CKGR_PLLAR_PLLACOUNT(0x3fUL) |
CKGR_PLLAR_DIVA(0x1UL))

    void SystemInit( void )
```

```
        /* Initialize main oscillator */
        PMC->CKGR_MOR |= CKGR_MOR_KEY_PASSWD | SYS_BOARD_OSCOUNT |
CKGR_MOR_MOSCXTEN;
        while ( !(PMC->PMC_SR & PMC_SR_MOSCXTS) );

        /* Switch to 3-20MHz Xtal oscillator */
        PMC->CKGR_MOR |= CKGR_MOR_KEY_PASSWD | CKGR_MOR_MOSCSEL;
        while ( !(PMC->PMC_SR & PMC_SR_MOSCSELS) );

        /* Initialize PLLA */
        PMC->CKGR_PLLAR = SYS_BOARD_PLLAR;
        while ( !(PMC->PMC_SR & PMC_SR_LOCKA) );

        /* Setting up prescaler */
        PMC->PMC_MCKR = PMC_MCKR_PRES_CLK_2 | PMC_MCKR_CSS_MAIN_CLK;
        while ( !(PMC->PMC_SR & PMC_SR_MCKRDY) ) ;

        /* Switch to PLLA */
        PMC->PMC_MCKR = PMC_MCKR_PRES_CLK_2 | PMC_MCKR_CSS_PLLA_CLK;
        while ( !(PMC->PMC_SR & PMC_SR_MCKRDY) ) ;
    }

    #ifdef __cplusplus
}
#endif
```

I/O initialization

To blink LEDs certain pins have to be configured as outputs, their voltage levels must change periodically.

There are several registers that control pins behaviour and they are grouped into PIO Controllers: PIOA, PIOB, etc. **[1, p.618]**. For them to operate properly we must enable their clocks **[1, p.528 - 28.7]**. I will enable all four PIO clocks although for now we need only PIOC and PIOA clock. TFT monitor connection will require usage of others PIOs too.

```
/* Activates clock to PIO controllers */
PMC->PMC_PCER0 =(1u << ID_PIOA) | (1u << ID_PIOB) | (1u << ID_PIOC) | (1u <<
ID_PIOD);
```

According to Arduino Due schematic, LED RX connected to PIOC line 30 and LED TX connected to PIOA line 21. See the following code for pins initialization, it is pretty self-explanatory and shows all necessary registers:

CPP

```
/* RX LED pin initialization (output, default high)*/
uint32_t Pin = 1u << (PIO_PC30_IDX & 0x1F);    //Moving 1 to position 30
PIOC->PIO_IDR = Pin;                           //No interrupt
PIOC->PIO_PUDR = Pin;                          //No pull-up
PIOC->PIO_MDDR = Pin;                          //No open collector
PIOC->PIO_CODR = Pin;                          //Low level
PIOC->PIO_OER = Pin;                           //Output
PIOC->PIO_PER = Pin;                           //Enables

/* TX Pin pin initialization (output, default high) */
Pin = 1u << (PIO_PA21_IDX & 0x1F);
PIOA->PIO_IDR = Pin;
PIOA->PIO_PUDR = Pin;
PIOA->PIO_MDDR = Pin;
PIOA->PIO_CODR = Pin;
PIOA->PIO_OER = Pin;
PIOA->PIO_PER = Pin;
```

I added this code into function SystemInit() to the bottom. All initialization code will go here.

Timer initialization

Having pins configured is not enough to blink LEDs, some kind of periodical event is needed in which LEDs are switched on and off. Timers can raise such even. The simplest timer to use

```
/* Initialization of SysTick */
/* Reload value is every 1ms. -1 is for 1 tick to reload value */
SysTick->LOAD = ((84000000/1000) & SysTick_LOAD_RELOAD_Msk) - 1;
/* Clearing current value     */
SysTick->VAL = 0;
/* Source is not divided by 8, enables SysTick, enables interrupt */
SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk |
SysTick_CTRL_TICKINT_Msk;
```

Again, I added this code into function SystemInit() to bottom after pins initialization.

As we can see, SysTick interrupt is enabled and will be fired every 1ms. That means it must be caught and served.

Catching SysTick interrupt

Now as pins are configured and there is an event that fires every 1ms, it is time to make LEDs blink. We won't be able to see blinking with 1kHz frequency, but 1Hz is fine. Thus we need a variable which will be increased by 1 every interrupt. Once it reaches 1000 - we toggle pins and zero the variable. Move to our main file, add the variable and interrupt handler. To know the name of interrupt handler I refer to startup_sam3xa.c file:

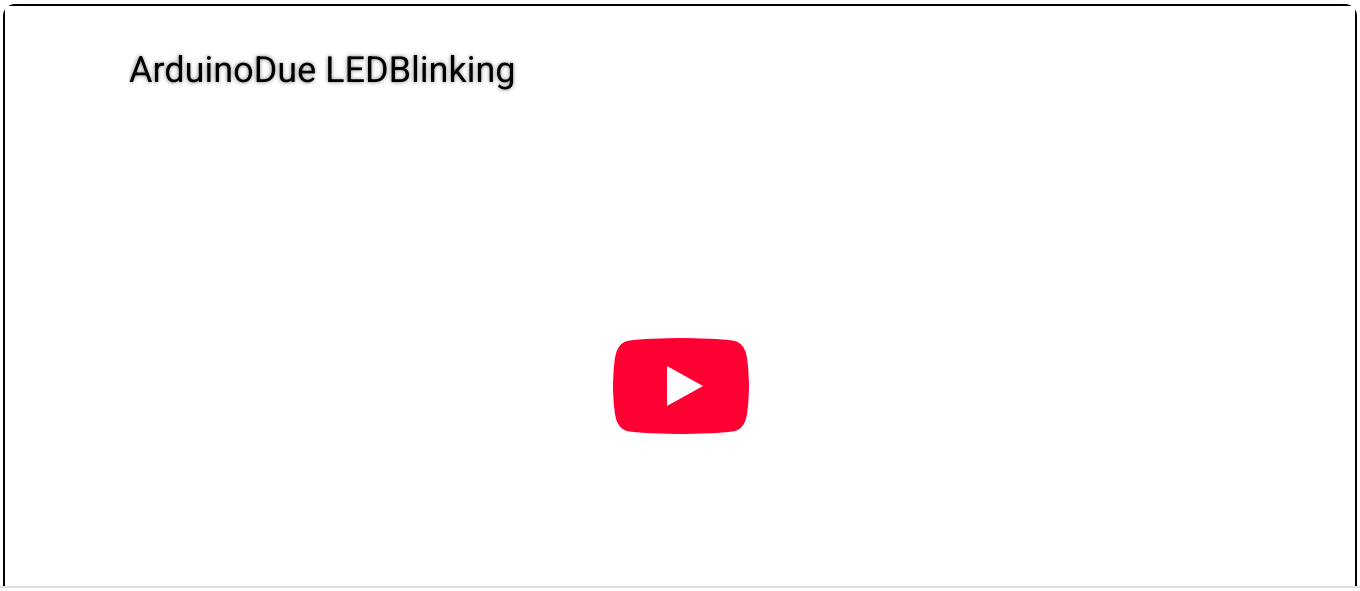
CPP

```
#include "sam.h"

uint32_t SysTickCounter;

void SysTick_Handler(void)
{
    uint32_t Pin_LEDRX;
    uint32_t Pin_LEDTX;

    SysTickCounter ++;
    if(1000 == SysTickCounter)
```

```
            if(PIOC->PIO_PDSR & Pin_LEDRX)
            {
                PIOC->PIO_CODR = Pin_LEDRX;    //Lights on

                PIOA->PIO_CODR = Pin_LEDTX;
            }
            else
            {
                PIOC->PIO_SODR = Pin_LEDRX;    //Lights off

                PIOA->PIO_SODR = Pin_LEDTX;
            }
        }
    }

    int main(void)
    {
        SystemInit();
        while(1);
    }
```

Compile it, upload to Arduino Due.

It should look like this:



Conclusion

In this article I shown some basics to start experimenting with Arduino Due not within the Arduino framework. In the next one I'll show how to initialize UART and how to print some messages back to your computer. I will create various print function for strings, number in hex, bin and decimal formats. Also I will initialize TFT monitor and show how to display something on it.

Source code is here.

Part two is here.

UPDATE 06-06-2015: Reloaded UVC 1.0 and UVC 1.0 uncompressed format specifications in zip (as pdfs are not allowed), updated links to USB 2.0 and SAM3X8E specifications as they have been moved to different locations. Page numbers in references to SAM3X8E reflect new edition of its specification.

UPDATE 10-07-2015: Reloaded source code without Debug folder.

License

by Sergiy Bogdancev
Contributor

Comments And Discussions (18)

Sort: Newest

Share your thoughts

You need to be signed in to participate in the discussion. Sign in

Anonymous

0 0 Reply

B

baykouchev

Oct 14, 2015

What about using the one descriptor for still image to capture one picture only? Is it that simple as select the descriptor and get only one video frame?

0 0 Reply

Hide reply

SB

Sergiy Bogdancev AUTHOR

Oct 14, 2015

Your camera's still image method is 2. Which means you can with the same end result just store the full frame, stop sending IN packets and send the whole video frame via RS232 to computer. Still image descriptor just tells you available frame sizes which in your case are the same as your video frame sizes. And thus, it comes to Arduino's memory size limitation - you only can store 176x144 max in gray-scale or 160x120 in color unless you connect some external memory (which you can, because you don't have TFT monitor occupying all connectors like in my case). And if you connect something like that: http://za.rs-online.com/web/p/fram-memory-chips/7165729/[^], you will probably be able to store even bigger frame sizes for still image. May be there are memory expansion shields for Arduino out there? The sending might take up to 10 sec. or so. I have not tried to send image from Arduino to computer, but I did send a bitmap from computer to Arduino via RS232, I could see how slow lines appeared on the Arduino's TFT monitor.

0 0 Reply

B

baykouchev

Oct 13, 2015

Great article and very experienced author. Thank you so much!

0 0 Reply

Hide reply

SB

Sergiy Bogdancev AUTHOR

Oct 13, 2015

Thanx. Did you come right with your camera so far? What's your progress?

baykouchev

B Oct 14, 2015

I added the watchdog reset you provided and now it's working fine with both Logging and Buffering. The only issue I had was mixed data from two video frames in the buffer. Because I don't use LCD display and I print all the bytes in ShowFrame to the Serial port. So my ShowFrame was very slow and it happens after reactivating USB communication, that I got just half a video frame. To fix it, I changed ShowFrame to this:

CPP

```
        //When frame is not fully stored (frame skipping)
        if(Index < VIDEO_FRAME_SIZE)  {
            Index = 0;
            return;
         }
```

Now each video frame starts at the beginning of the buffer.

0 0 Reply

B baykouchev
Sep 27, 2015

Hi Sergiy, well done! It's incredible what you have given to the community. I've build your project and connected with my cam - everything working fine so far. The only strange thing is the frame data. It looks like this:

TEXT

```
0 T:  12 H:12 BF:1000 1100 F: 726-4 BANK0
1 T:  12 H:12 BF:1000 1100 F: 726-5 BANK1
2 T:  12 H:12 BF:1000 1100 F: 726-6 BANK2
3 T:  12 H:12 BF:1000 1100 F: 726-7 BANK0
4 T:  12 H:12 BF:1000 1100 F: 727-0 BANK1
5 T:  12 H:12 BF:1000 1100 F: 727-1 BANK2
...
```

regardless what I set EP_SIZE 1, 2, 3 etc. Do you have an idea what could be the reason, that total size = header size = 12? Seems no data is sent, but frames are counted, strange... BR

SB

Sergiy Bogdancev AUTHOR
Sep 27, 2015

Hi Anton I'm pleased that someone tries it too - it is very interesting exercise. First of all your camera is responding which is good. Second, for how long did you run the logging? Did you try something like 3000 entries? Are they all only 12 bytes headers? Edit: it is not frames counted (I mean not video frames) - rather transactions counted (or USB micro-frames). It might be that camera is not ready to send data yet, it will send empty headers for some time and only then starts sending headers + payload.

0 0 Reply

SB

Sergiy Bogdancev AUTHOR
Sep 28, 2015

I can now see where it could go wrong. In my camera isochronous endpoint has number 2, in your camera - it is number 1. See one of the descriptors (I made it bold and underlined):

```
-------------------------------------
Endpoint Descriptor:
bLength: 7
bDescriptorType:  0x05
bEndpointAddress: 1000 0001
bmAttributes:     0000 0101
wMaxPacketSize:   0000 0000 1000 0000
bInterval:        1
-------------------------------------
```

so now find following function in USBD.c and change 2 to 1 where IN pipe initialized:

CODE

```
void USBD_SetInterfaceAlternateSettingEnd(uint16_t
ByteReceived)
{
    PrintStr("Alternate setting has been set.\r\n");
    if(HCD_InitiateIsochronousINPipeOne(DeviceAddress, 2))
    {
        //...
}
```


0 0 Reply

Hide reply

B

baykouchev
Sep 28, 2015

Hi Sergiy, That is something I've changed already. Without it, no data packages are sent at all. I also changed frame_index according to my cam. My camera has no audio. Regards, Anton

0 0 Reply

SB

Sergiy Bogdancev AUTHOR
Sep 28, 2015

Transactions log you sent me lasts about 375ms = 1101-726, on some forums about drivers I read that their cameras produce just headers for about 800ms and only then start producing header + payload. My camera starts a bit faster. The question is: Do you have screen attached or you're just logging? Because if you just logging - try to log a lot more or shift it in time. Currently because I did not set interrupt priorities Arduino will reset if RS232 output takes too long. The solution would be to alter logging method to skip first 5000 or more transaction and then start logging or disable Watchdog timer at the beginning so it does not expire during longer logging (would you be able to do one of those?).

0 0 Reply

SB

Sergiy Bogdancev AUTHOR
Sep 28, 2015

indeed if I skip first 5000 calls of VP_ProcessPartialPayload_Logging and start Monitoring next 3000 I (see the trace below with 512 data buffer). Does it means my camera is to slow even with 176x144 screen? Can I speed up it? No, it just means your camera start outputting data with bigger initial delay. It does not affect further operation. **Do see it right, that VP_ProcessPartialPayload_Buffering or _Logging are called with some frequency independent from data being sent from the cam? Where can this frequency be controlled?** They are called every USB micro-frame, in other words every 0.125ms whether camera has a data or not. You don't need to control that because you don't know when camera will have a data. I do stop it (IN packet generation) when buffer has a full video frame and resume generation when video frame has been output to screen. **Another point is trace always ends with "VBus error." and Arduino Due restarts. Can I let it run forever?** Normal operation MUST run forever, but logging cannot because Arduino has defined RAM size that you can use to store transaction data. Also, did you fix Watchdog problem during long RS232 data

176x144 resolution well in gray-scale. It would be nice to try ATMEL's Cortex-M7 with

300MHz clock, that should do well even with 640x480 in gray-scale and lower than that in color. With no to little code alteration.

0 0 Reply

Hide reply

B

baykouchev
Sep 29, 2015

thanks alot for the response. Now I get video frames but also empty (header only) USB micro-frames in between, which is OK according to your explanation. I think now I got the idea with the USB micro-frames. Do you also get empty USB micro-frames between video frames with your camera? Only think I don't know is how to fix Watchdog problem with RS232. Looks like it happens as you say - after first output of 3000 USB micro-frames MCU get restarted. You mentioned something about interrupts priority?

0 0 Reply

Hide replies

SB

Sergiy Bogdancev AUTHOR
Sep 29, 2015

I think now I got the idea with the USB micro-frames. Do you also get empty USB micro-frames between video frames with your camera? Yes, I do. I have them in between video-frames and even in between transactions inside a video-frame. See article #8, there are several screenshots about "pauses". Think of that time when you are getting empty packets as a time to process something (to send data to screen for example). If I output in color, I would also use that time to convert YUY2 into RGB, etc. **Only think I don't know is how to fix Watchdog problem with RS232. Looks like it happens as you say. After first output of 3000 USB micro-frames MCU get restarted.** Watchdog timer is re-loaded every second in function void SysTick_Handler(void) in WebcameraCapturing.c file. If for some reason it does not happen for several seconds (8 sec if I'm not mistaken) - MCU resets. You can copy

CODE

```
WDT->WDT_CR = WDT->WDT_CR | WDT_CR_KEY_PASSWD |
```

CODE

```
void VP_PrintHeaderInfo(uint32_t Count)
```

function inside **for()** clause so it reloads it every time (not nice but should work). As alternative, you can disable Watchdog timer at the beginning, according to SAM3X datasheet: *After a Processor Reset, the value of WDV is 0xFFF, corresponding to the maximum value of the counter with the external reset generation enabled (field WDRSTEN at 1 after a Backup Reset). This means that a default Watchdog is running at reset, i.e., at power-up. The user must either disable it (by setting the WDDIS bit in WDT_MR) if he does not expect to use it or must reprogram it to meet the maximum Watchdog period the application requires.* **You mentioned something about interrupts priority?** This is a big topic, basically you can assign priorities to interrupts so if higher priority interrupt happens, it preempts currently running interrupt. As I did not assign any priorities, I assume that SysTick and USART Output ready interrupts have the same priority. When USART sends, it constantly in interrupt handling mode, so SysTick interrupt is pending and as it is pending - watchdog counts to 0 and reset happens.

0 0 Reply

SB

Sergiy Bogdancev AUTHOR
Sep 29, 2015

As an additional note about term "frame". There are 3 things that have the same word "frame" in their names. Other words cannot be used because that is how they are in documentation. And I can see it confuses you. 1. **Video-Frame** is a bunch of pixels (one pixel can be 1 or more bytes) that form a "still picture", you need periodically change those video-frames to make it "motion picture". 2. **USB micro-frame** is an interval between two SOF (start-of-frame) packets. SOF consists of defined by USB standard token. Any other packets happen inside micro-frames and cannot last longer than micro-frame, i.e. they cannot cross SOF boundaries. When USB speed is HS (high speed), SOFs appear every 0.125ms. 3. **USB frame** is 8 (eight) USB micro-frames and thus it happens every 1ms. It is just a name, it has no physical meaning because there are no special tokens or packets for that. You can group any 8 adjacent micro-frames and call it a frame. However,

CODE

```
0 T:  12 H:12 BF:1000 1100 F: 726-4 BANK0
1 T:  12 H:12 BF:1000 1100 F: 726-5 BANK1
2 T:  12 H:12 BF:1000 1100 F: 726-6 BANK2
3 T:  12 H:12 BF:1000 1100 F: 726-7 BANK0
4 T:  12 H:12 BF:1000 1100 F: 727-0 BANK1
5 T:  12 H:12 BF:1000 1100 F: 727-1 BANK2
...
```

Field F, first number is frame, second after "-" is one of eight micro-frames that compose given frame. Thus the correct explanation of the whole process is: you program SAM3X's USB module to produce isochronous transfers to your camera. Isochronous (periodic) transfer unlike control transfer consists of one transaction only that happens once every USB **micro-frame** (it is also possible to do several transactions inside one micro-frame but SAM3X won't handle that out of the box). Each transaction consist of IN token from host and optional DATA packet from camera. If DATA packet is returned by camera, it can have data or be empty. If it has data, it can be just header or header with payload. If it has payload, the payload will have partial pixel data of a **video-frame**. So, you need to get many DATA packets with payload to compose full **video-frame**. In our case: **SOF** - [IN - DATA] - **SOF** - [IN - DATA] - **SOF** - [IN - DATA] ... For faster processors and bigger video-frames: **SOF**

0 0 Reply

P

Phebous
Jul 13, 2015

This is the most complex hello world example I have see for Arduino. You have my complements!

0 0 Reply

B

BillW33
Mar 17, 2015

Interesting and well written.
Just because the code works, it doesn't mean that it is good code

Great article! It seems most everyone used the Arduino IDE it's nice to see other using Atmel Studio. I use AS exclusively and it's a bear to find code examples. The Due is a great device! New version: WinHeist Version 2.1.0 My goal in life is to have a psychiatric disorder named after me. I'm currently unsupervised, I know it freaks me out too but the possibilities are endless.

0 0 Reply