

XCAM N329- SDK Manual

|  |  |
| --- | --- |
| Version: | 1.8 |
| Date: | 5th Dec 2018 |
| Author: | LAU Yat Fei Jackson |

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Comment** |
| 1.0 | 5 Dec 2017 | Original release |
| 1.1 | 1 Feb 2018 | UART Interface + Multiple Colour Palette + POIs |
| 1.2 | 22 March 2018 | Support dual camera and adaptive colour palette |
| 1.3 | 14 May 2018 | Update Colour palettes |
| 1.4 | 17 July 2018 | Update UVC temperature display  Temperature data padding bits increase to 13 |
| 1.5 | 1 Nov 2018 | Update UART commands |
| 1.6 | 16 Nov 2018 | Create page “Frame Rate”  Add operation guideline of updating firmware |
| 1.7 | 23 Nov 2018 | Add Temperature Colour Mapping section |
| 1.8 | 5 Dec 2018 | Add APIs(DeadPixelMasking, N320\_VideoStreaming)  Create “Dead Pixel Masking” |
|  |  |  |
|  |  |  |

Contents

XCAM Foundation 3

Library Structure 3

Single camera module 3

Dual camera module 4

Project Setting 5

Header 5

Software dependencies 5

Preprocessor Symbols 5

Frame Rate 6

ThermalSensorAPI\_N329.lib 7

XCAM Constant 8

Customised data structure 8

API 11

Temperature Colour Mapping 18

Data Padding 20

Dead Pixels Masking 21

UART/HUART Interface 23

Settings 23

Output commands 23

Build and Test 25

Update Firmware 25

Demonstration 27

Single camera module 27

Dual camera module 27

Resources 28

# XCAM Foundation

The documentation of the XCAM SDK for users.

The XCAM SDK basically contains fundamental functions which supports for MCU **N329-** series library to control XCAM. Dual camera module is using additional library, below is the main project structure of XCAM\_N329.

## Library Structure

### Single camera module

|----N32901-3\_Non-OS\_KEIL

|----I2C

|----lib/w55fa93\_i2c.h

|----XCam\_Single

|----example/SwLib3

|---- I2CS/DrvI2C\_Thermal.c

|---- I2CS/DrvI2C.h

|---- main.c

|---- standalone.c

|---- Table\_UVC.c

|---- usbd\_video.c

|--- -wb\_init.s

|---- KEIL

|----MI\_Thermal.uvproj

|----MI\_Thermal/MI\_XCAM\_N329\_SW\_I2C.bin

|----lib/videoclass\_HTPA32.h

|----lib/w55fa93\_UVC\_YUB\_Only.lib

|----N32901

|----N32901.sct

|----SYSLIB/Lib

|----w55fa93\_reg.h

|----W55FA93\_syslib.lib

|----wberrcode.h

|----wbio.h

|----wblib.h

|----wbtypes.h

|----ThermalAPI\_inC

|----lib/videoclass\_HTPA32.h

|----example/demo/KEIL/ThermalSensor.h

|----example/demo/KEIL/MI\_XCAM.h

|----example/demo/KEIL/demo\_Data/N32901\_SingleTask/ThermalSensorAPI\_N329.lib

|----UDC

|----lib/usbd.h

|----lib/w55fa93\_UDC.lib

|----VPOST

|----lib/w55fa93\_vpost.h

|----lib/W55FA93\_VPOST.lib

### Dual camera module

|----N32901-3\_Non-OS\_KEIL

|----XCAM\_Dual

|----lib/videoclass\_HTPA32.h

|----lib/w55fa93\_UVC\_YUB\_Only.lib

|----example/HTPA32

|---- I2CS

|---- DrvI2C\_Thermal.c

|---- DrvI2C.c

|---- DrvI2C.h

|---- VIN

|---- GCD.c

|---- Smpl\_FSC\_Ov7740.c

|---- Smpl\_I2C.c

|---- VIN\_demo.c

|---- vin\_demo.h

|---- main.c

|---- standalone.c

|---- Table\_UVC.c

|---- usbd\_video.c

|--- -wb\_init.s

|---- KEIL

|----MI\_Thermal.uvproj

|----MI\_Thermal/MI\_XCAM\_N329\_SW\_I2C.bin

|----gpio

|----lib/w55fa93\_gpio.h

|----lib/w55fa93\_gpio.lib

|----N32901

|----N32901.sct

|----SYSLIB/Lib

|----w55fa93\_reg.h

|----W55FA93\_syslib.lib

|----wberrcode.h

|----wbio.h

|----wblib.h

|----wbtypes.h

|----ThermalAPI\_inC

|----lib/videoclass\_HTPA32.h

|----example/demo/KEIL/ThermalSensor.h

|----example/demo/KEIL/MI\_XCAM.h

|----example/demo/KEIL/demo\_Data/N32901\_SingleTask/ThermalSensorAPI\_N329.lib

|----UDC

|----lib/usbd.h

|----lib/w55fa93\_UDC.lib

|----VideoIn

|----lib/W55fa93\_VideoIn.h

|----lib/W55FA93\_VideoIn.lib

|----VPOST

|----lib/w55fa93\_vpost.h

|----lib/W55FA93\_VPOST.lib

**SDK Version: Release V3.08 181205**

## Project Setting

### **Header**

"ThermalSensor.h”

“w55fa93\_vpost.h”

“videoclass\_HTPA32.h”

### **Software dependencies**

..\..\..\lib

..\..\..\..\SYSLIB\Lib

..\..\..\..\I2C\lib

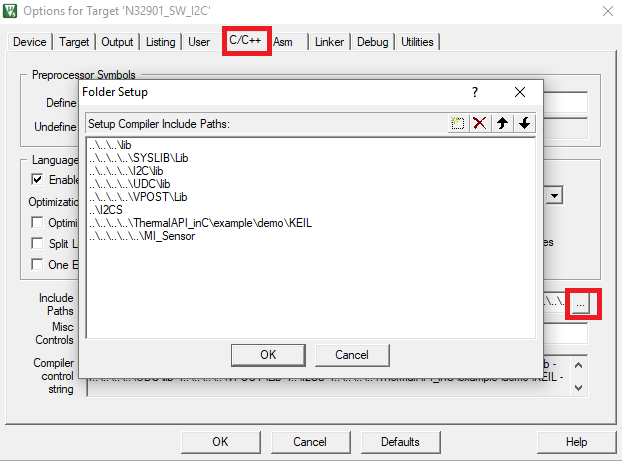
..\..\..\..\UDC\lib

..\..\..\..\VPOST\Lib

..\I2CS

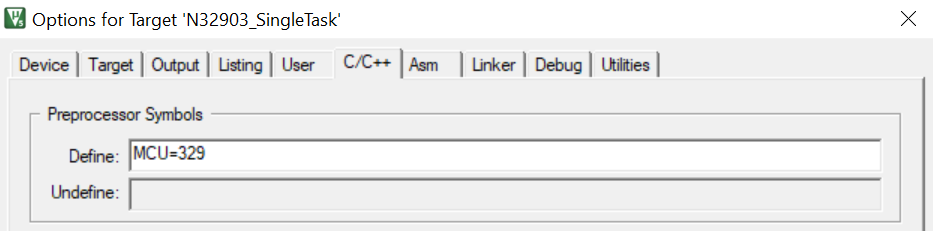
..\..\..\..\ThermalAPI\_inC\example\demo\KEIL

..\..\..\..\..\MI\_Sensor

****

### **Preprocessor Symbols**

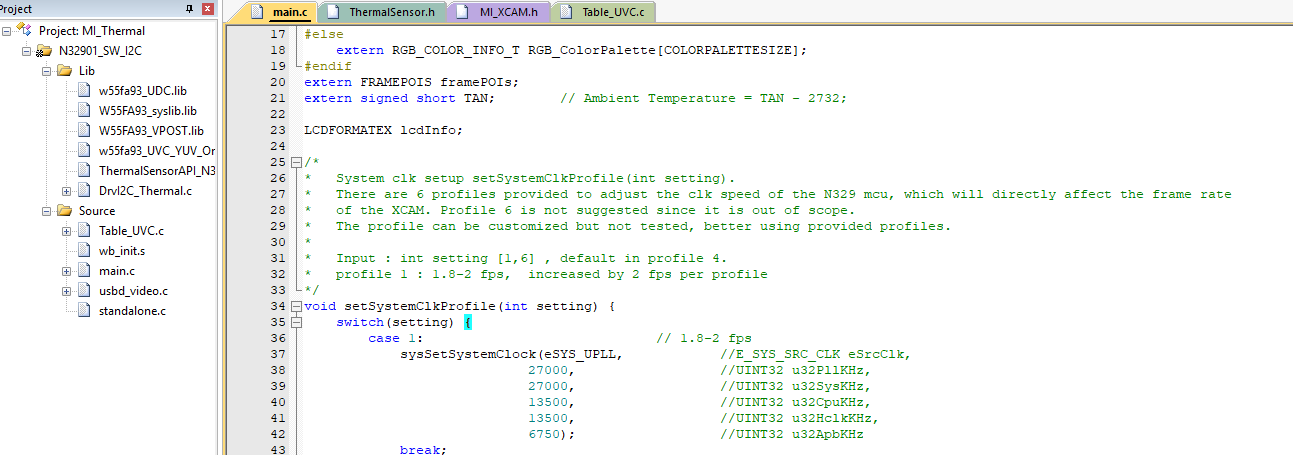
Define: MCU=329



### **Frame Rate**

XCAM can run at 1 – 8 fps without affecting performance, depending to the profile setting of system clock. It is advised to use setting provided at setSystemClkProfile() in main.c.

Changing the system clock speed will affect the frame rate, also the current and power consumptions. It is highly recommended to choose appropriate clock speed for different purpose.

****

If user needs customized profile, please contact software engineer for further support. Incorrect clock speed setting leads to abnormal behaviour.

### **ThermalSensorAPI\_N329.lib**

#define DEADPIXELCOMPENSATE

Usage: Dead pixel(s) compensation using average masking.

Prerequisite: All the dead pixels should be stored at EEPROM or deadPixelList[] that includes:

1) Number of dead pixel and

2) Coordinates of dead pixels

\*If there are any dead pixels next to each other, pay attention to use correct mask

\*Pre-set from ThermalSensorAPI\_N329.lib

#define POI

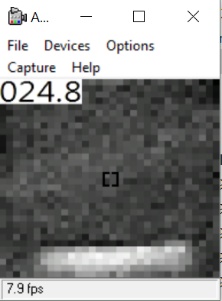
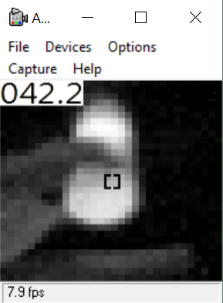
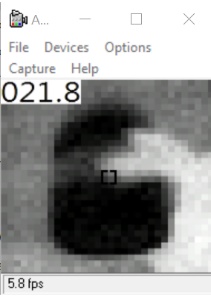
Usage: Point of interests (POIs) recording for each frame. If not define, then FRAMEPOIS object will be empty.

\*Pre-set from ThermalSensorAPI\_N329.lib

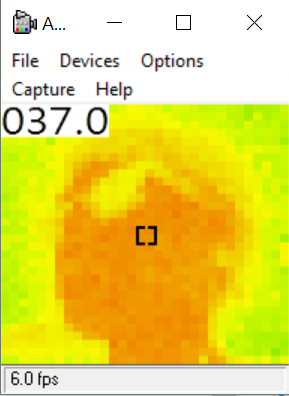
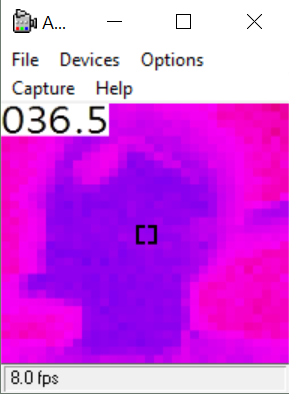
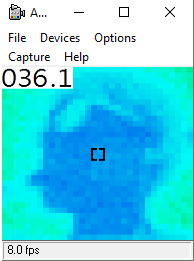
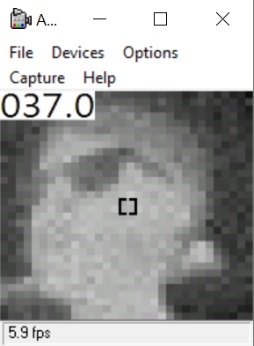
#define COLOR\_ADAPTIVE

Usage: The FRAMEPOIS determine how the temperature spread on colour palette

Better to use colour table under wide range tag, i.e.: COLORPALETTE0\_WIDERANGE

#define COLORPALETTExxxxx…

Usage: Define colour palette used

#define MIN\_MAX\_TEMP\_DISPLAY

Usage: Display only average temperature or max/min temperature on screen

Implementation can be customized in main.c🡪TempDisplay()

### **XCAM Constant**

|  |  |  |
| --- | --- | --- |
| **Variables** | **Values** | **Usage** |
| WIDTH | 32 | Width of the sensor |
| HEIGHT | 32 | Height of the sensor |
| W\_WIDTH | 2 | Width of temperature calculation window |
| W\_HEIGHT | 2 | Height of temperature calculation window |
| VERSION | yymmdd | SDK version |
| COLORTABLESIZE | 1200 | Size of color table, support in range of 0.0 – 120.0 degrees mapping |

### **Customised data structure**

|  |
| --- |
| peri\_interface : Peripheral interface |
| enum peri\_interface{  HUART = 0,  UART,  SPI }; |

|  |
| --- |
| YUV\_COLOR\_INFO\_T: YUV data structure |
| typedef struct YUV\_COLOR\_INFO\_T {  unsigned int YUVData;  } YUV\_COLOR\_INFO\_T; |

|  |
| --- |
| RGB\_COLOR\_INFO\_T: RGB data structure |
| typedef struct RGB\_COLOR\_INFO\_T{  unsigned char R;  unsigned char G;  unsigned char B;  } RGB\_COLOR\_INFO\_T; |

|  |
| --- |
| REGISTERSETTING : Data structure storing the values of sensor register. |
| typedef struct REGISTERSETTING{  unsigned short MBIT;  unsigned short BIAS;  unsigned short CLOCK;  unsigned short BPA;  unsigned short PU;  } REGISTERSETTING; |

|  |
| --- |
| EEPROM: Data structure storing values read from EEPROM. |
| typedef struct EEPROM{  float PixCMin; // Minimum sensitivity coefficient, used for scaling  float PixCMax; // Maximum sensitivity coefficient, used for scaling  unsigned short gradScale; // Emissivity factor  unsigned short TableNumberSensor; // The look-up table number of sensor belongs  unsigned short epsilon; // Factor for fine tuning of the sensitivity for all Pixel  REGISTERSETTING calibRegister; // Sensor register values for calibration  unsigned short VddRef; // used supply voltage during calibration measured  float PTATGrad; // Factor of calculating ambient temperature (Ta)  float PTATOff; // Factor of calculating ambient temperature (Ta)  unsigned char VddScaling; // VddComp scaling coefficient  unsigned short VddScalingOff; // VddComp scaling coefficient  unsigned char GlobalOffset; // Factor for fine tuning of the sensitivity for all Pixel  unsigned short GlobalGain; // Factor for fine tuning of the sensitivity for all Pixel  REGISTERSETTING userRegister; // Sensor register values for user  unsigned short DevID; // Device ID  unsigned char NrOfDefPix; // Number of dead pixel(s)  unsigned short DeadPixAdr[MAXNROFDEFECTS]; // Array of dead pixel addr  unsigned char DeadPixMask[MAXNROFDEFECTS]; // Array of dead pixel mask  signed short VddGrad[ELAMOUNT]; // VddComp gradient  signed short VddOff[ELAMOUNT]; // VddComp offset  signed short ThGradN[Pixel]; // thermal gradient  signed short ThOffN[Pixel]; // compensate for any thermal drifts  unsigned long PixCN[Pixel]; // Sensitivity coefficients  } EEPROM; |

|  |
| --- |
| SENSORSETTING: Data structure storing sensor look-up table values for TO calculation. |
| typedef struct SENSORSETTING{  signed long TABLENUMBER;  signed long long PCSCALEVAL; // defined scaling coefficient  signed long NROFTAELEMENTS;  signed long NROFADELEMENTS;  signed long TAEQUIDISTANCE;  signed long ADEQUIDISTANCE;  signed long ADEXPBITS;  signed long TABLEOFFSET;  unsigned char MBITTRIMDefault;  signed long SensRv;  unsigned int\* TempTable;  unsigned int\* XTATemps;  unsigned int\* YADValues;  } SENSORSETTING; |

|  |
| --- |
| TEMPIXEL: POI information |
| typedef struct TEMPIXEL{  unsigned short x;  unsigned short y;  signed short Tmp;  } TEMPIXEL; |

|  |
| --- |
| DEADPIXEL: Dead pixel information |
| typedef struct DEADPIXEL {  unsigned short x;  unsigned short y;  int mask;  } TEMPIXEL; |

|  |
| --- |
| FRAMEPOI: Storing POIs of single frame |
| typedef struct FRAMEPOI{  TEMPIXEL maxTemPixel;  TEMPIXEL maxTemPixel;  } FRAMEPOIS; |

|  |
| --- |
| DEADPIXEL\_LIST: Storing all additional dead pixels information defined in main.c deadPixelList[] |
| typedef struct DEADPIXEL\_LIST{  DEADPIXEL\* pDeadPixels;  int numOfElement;  } FRAMEPOIS; |

### **API**

**ThermalSensor.h**

void N329\_InitSensor(void);

void N329\_OpenSensor(void);

void N329\_VideoStreaming(void);

int N329\_Interface\_init (unsigned short interface);

void N329\_UartDataValid\_Handler (UINT8\* buf, UINT32 u32Len);

**MI\_XCAM.h**

void InitI2C(unsigned char mode);

void HighDensSequentialRead(unsigned short address,unsigned char\* data, unsigned short numbytes);

void HighDensPageWrite(unsigned short address,unsigned char\* data, unsigned short numbytes);

void ReadCalibDataN(void);

void InitSensorDev(unsigned short TN);

void InitMBITTRIMN(unsigned char user);

void InitBIASTRIMN(unsigned char user);

void InitBPATRIMN(unsigned char user);

void InitPUTRIMN(unsigned char user);

void InitCLKTRIMN(unsigned char user);

unsigned int CalcTO(unsigned int TAmb, signed int dig, signed long PiC, unsigned int dontCalcTA);

void Create\_color\_table(RGB\_COLOR\_INFO\_T RGB\_ColorPalette[],YUV\_COLOR\_INFO\_T YUV\_ColorTable[]);

void DeadPixelMasking (DEADPIXEL pixel);

FRAMEPOIS GetFramePOIs (void);

void GetImageData (void);

int GetTargetPixelIndex(void);

unsigned short GetTemp(unsigned int x, unsigned int y);

unsigned short GetTempDisplay(void);

void ResetFramePOIs(void);

void SetTargetPixelIndex(int index);

void SetTempDisplay(unsigned short flag);

unsigned int StartStreaming(int Mode, char Temps, char Stream);

|  |  |
| --- | --- |
| void | N329\_InitSensor(void); |



Open and setup the system environment for I2C.

|  |  |
| --- | --- |
| void | N329\_Open(void); |

Initialization of sensor, mainly read calibration data and setup trim registers.

|  |  |
| --- | --- |
| int | N329\_Interface\_init (unsigned short input); |



Initialization of peripheral interface, input HUART/UART as parameter.

|  |  |
| --- | --- |
| void | N329\_VideoStreaming(void); |

Contains video streaming and invoke UVC event.

|  |  |
| --- | --- |
| void | N329\_UartDataValid\_Handler (void); |

Handle the UART/HUART command list.

|  |  |
| --- | --- |
| void | InitI2C(unsigned char mode); |

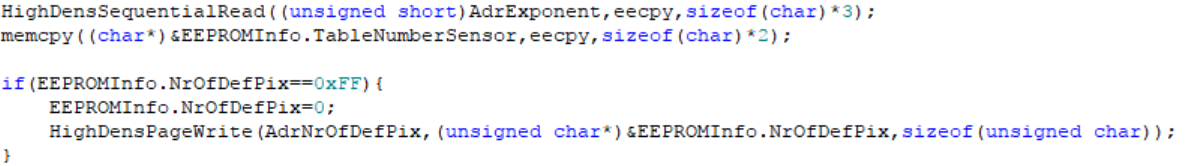


Setup the EEPROM & Sensor address to the device.

Dependencies: 0🡪 Init I²C for Sensor (> 1000 kHz)

1🡪 Init I²C for EEPROM (max 400 kHz)

|  |  |
| --- | --- |
| void | HighDensSequentialRead(unsigned short address,unsigned char\* data, unsigned short numbytes); |
| void | HighDensPageWrite(unsigned short address,unsigned char \*data, unsigned short numbytes); |



HighDensSequentialRead() & HighDensPageWrite() reads/writes multiple bytes from a high-density (>= 32 Kb) serial EEPROM device.

Dependencies: 'address' contains address word

'data' contains the reading result

'numbytes' contains the length of bytes to read

|  |  |
| --- | --- |
| void | ReadCalibDataN(void); |



Read all required calibration data from sensor EEPROM, details in TYPEDEF structure EEPROM.

|  |  |
| --- | --- |
| void | InitSensorDev (unsigned short TN); |



Open and setup the sensor device, pick up the setting corresponding to the type of optics by TableNumber.

|  |  |
| --- | --- |
| void | InitMBITTRIMN(unsigned char user); |



Dependencies: 0🡪 Setting during Calibration

1🡪 Setting from user

Range: 4 <= m <= 12

Initialization of Trim Register 1 MBIT/ (m+4) bit as ADC resolution to sensor register.

|  |  |
| --- | --- |
| void | InitBIASTRIMN(unsigned char user); |



Dependencies: 0🡪 Setting during Calibration

1🡪 Setting from user

Range: 0 to 31 🡺 1μA to 13μA

Initialization of Trim Register 2, adjust the bias current of the ADC.

|  |  |
| --- | --- |
| void | InitBPATRIMN(unsigned char user); |



Dependencies: 0🡪 Setting during Calibration

1🡪 Setting from user

Range: 0 to 31 🡺 0.2μA to 4.0μA

Initialization of Trim Register 5, adjust the common mode voltage of the preamplifier.

|  |  |
| --- | --- |
| void | InitPUTRIMN(unsigned char user); |



Dependencies: 0🡪 Setting during Calibration

1🡪 Setting om user

Range: “1000” = 100 kOhm; “0100” = 50 kOhm; “0010” = 10 kOhm; “0001” = 1 kOhm

Initialization of Trim Register 7, select internal pull up resistor on SDA/SCL.

|  |  |
| --- | --- |
| void | InitCLKTRIMN(unsigned char user); |



Dependencies: 0🡪 Setting during Calibration

1🡪 Setting from user

Range: 0 to 63 🡺 1MHz to 13MHz

Initialization of Trim Register 4, clock frequency setting CLK\_TRM.

|  |  |
| --- | --- |
| unsigned int | CalcTO(unsigned int TAmb, signed int dig, signed long PiC, unsigned int dontCalcTA); |



CalcTO() calculate the object temperature via look-up table.

Dependencies: TAmb = ambient temperature

dig = pixel voltage

PiC = pixel sensitivity coefficients

Return: Object Temperature in dK

|  |  |
| --- | --- |
| void | Create\_color\_table(RGB\_COLOR\_INFO\_T RGB\_ColorPalette[], YUV\_COLOR\_INFO\_T YUV\_ColorTable[]); |

create\_color\_table () create color table and data for transfer.

To scale RGB palette size to YUV table size, the color between two specified consecutive color in RGB palette is interpolated with the following equation

c = a+(b-a)\*t,

while c = interpolated result, a is one of the color element in RGB palette, b is consecutive next color element in palette, and t = the interpolation size (ceil(YUV table size /RGB palette size)) times the order of element in the interpolation. Interpolated RGB values is calculated using above equation.

Next, the RGB is converted to YUV based on the following equations:

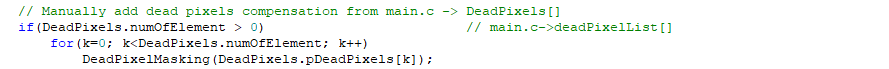
**Y = R \* .299 + G \* .587 + B \* .114;  
U = R \* -.169 + G \* -.332 + B \* .500 + 128.;  
V = R \* .500 + G \* -.419 + B \* -.0813 + 128.;**

Dependencies: RGB\_ColorPalette[] = chosen color palette

YUV\_ColorTable = converted YUV color table from RGB

Return: /

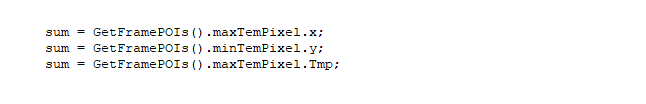
|  |  |
| --- | --- |
| void | DeadPixelMasking(DEADPIXEL pixel); |



Do averaging for input DEADPIXEL data structure.

Return: /

|  |  |
| --- | --- |
| FRAMEPOIS | GetFramePOIs(void); |



GetFramePOIs return the FRAMEPOIS object.

Return: data structure FRAMEPOIS

|  |  |
| --- | --- |
| Void | GetImageData(void); |



GetImageData() calculate voltage values of each pixels.

|  |  |
| --- | --- |
| unsigned short | GetTargetPixelIndex(void); |

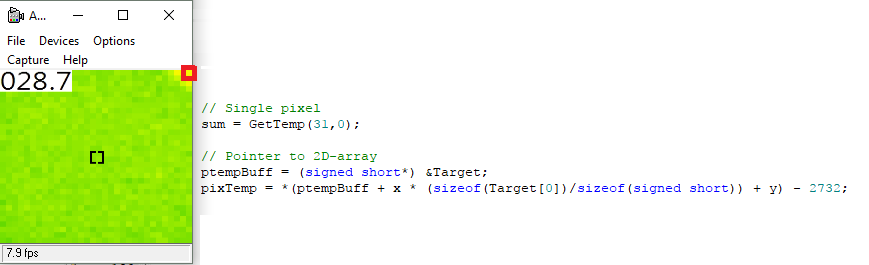


GetTargetPixelIndex() set which pixel temperature should be shown.

Dependency: /

Return: The temperature value of targeted pixel.

|  |  |
| --- | --- |
| unsigned short | GetTemp(unsigned int x, unsigned int y); |



GetTemp(x,y) gets the calculated temperature from 2D-array sensor, in single pixel.

\* P.S: If you want to get the whole 2D-array, you may use

ptempBuff = (signed short\*) &Target;

pixTemp = \*(ptempBuff + x \* (sizeof(Target[0])/sizeof(signed short)) + y) - 2732;

Dependencies: unsigned int x= x-coordinate of target pixel (0 < x < COLUMN-1)

unsigned int y = y-coordinate of target pixel (0 < y < ROW-1)

Return Code: Celsius °C x 10 (e.g: 301 = 30.1°C, right shifted 1 decimal)

998 Coordinate input ERROR

|  |  |
| --- | --- |
| unsigned short | GetTempDisplay(void); |



GetTempDisplay() get the temperature display flag. The flag is set by SetTempDisplay().

Dependency: /

Return: The flag of showing bracket and temperature by UVC graphic.

|  |  |
| --- | --- |
| void | ResetFramePOIs(void); |

ResetFramePOIs() reset the FRAMEPOIS object.

|  |  |
| --- | --- |
| void | SetTargetPixelIndex(int index); |

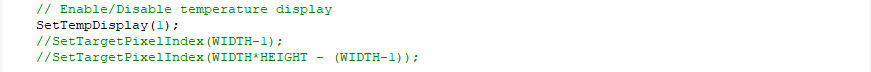
SetTargetPixelIndex() set which pixel we want to show its temperature on the display.

Dependencies: 0 <= index <= WIDTH\*HEIGHT

-1(Default) Average temperature of bracket area

Return value: /

|  |  |
| --- | --- |
| void | SetTempDisplay(unsigned short flag); |



SetTempDisplay() set to turn on/off of showing temperature.

Dependencies: flag = On(1) or Off(0)

|  |  |
| --- | --- |
| unsigned int | StartStreaming(int Mode, char Temps, char Stream); |



StartStreaming() starts the streaming of the sensor, main sequence can be seen in the interrupt.

Dependencies: Mode = Initialize(1) or Normal streaming (0)

Temps: whether it is calculating temperature

Stream = whether it is streaming mode

Return: 0x0 / 0xFF

## Temperature Colour Mapping

Based on the chosen RGB colour palette, it will first expand to the YUV colour table with self-defined number of elements in *Create\_color\_table()*. Then, it can map the temperature to corresponding YUV value by entry index. There are 2 methods for the colour mapping.

Now we have colour table with fixed 1200 elements.

**[0 ≤ Index < 1200]**

For adaptive colour mapping, , where 1199 is the index range of the table start from element 0, it is varied depends on the max & min value on each frame.

The Index is equal to the temperature value.

If maxT = 20°C, minT = 30°C in particular frame, and temp of the point of interest[i][j] = 25°C with RGB value {R0,G0,B0}, . Thus, the 599th colour code on the RGB colour table shown on the screen to represent 25°C. In alternative, adaptive mode is evenly distributing the whole colour palette to valid temperature on frame. Pixel[i][j] will be mapped to:

* RGB\_ColorTable[599] = {R0,G0,B0} <-> {Y0,U0,V0}

For non-adaptive colour mapping, the Index is equal to the temperature value right shifting 1 digit plus arbitrary colour offset (600). The shifting is applied in order to support 1 decimal value without using float data type. The offset is used for adjusting the pixel colour falling to certain range of palette.

If temperature of first pixel is 25.2°C, the index becomes 25.2\*10 + 600 = 852

* RGB\_ColorTable[852] = {R1,G1,B1} <-> {Y1,U1,V1}

According to the relationship of temperature data and corresponding YUV value, we can do reverse calculation on application software to get back temperature data. For more details please refers to [rgb2temp](#_Resources) example.

void Create\_color\_table(RGB\_COLOR\_INFO\_T RGB\_ColorPalette[],YUV\_COLOR\_INFO\_T YUV\_ColorTable[])

{

int i,j,interpolateSize;

unsigned char RVal,GVal,BVal, Color\_Y, Color\_U, Color\_V;

interpolateSize = COLORTABLESIZE/(COLORPALETTESIZE-1); //1200/60

for(i=0; i< (COLORPALETTESIZE-1); i++)

{

for(j=0;j<interpolateSize;j++)

{

// Here is expanded RGB Color Table

RVal = (unsigned char)((float)RGB\_ColorPalette[i].R + ((float)RGB\_ColorPalette[i+1].R - (float)RGB\_ColorPalette[i].R) /interpolateSize\*j);

GVal = (unsigned char)((float)RGB\_ColorPalette[i].G + ((float)RGB\_ColorPalette[i+1].G - (float)RGB\_ColorPalette[i].G) /interpolateSize\*j);

BVal = (unsigned char)((float)RGB\_ColorPalette[i].B + ((float)RGB\_ColorPalette[i+1].B - (float)RGB\_ColorPalette[i].B) /interpolateSize\*j);

// RGB2YUV

Color\_Y = (unsigned char)(RVal \* 0.299 + GVal \* 0.587 + BVal \* 0.114);

Color\_U = (unsigned char)(RVal \* (-0.169) - GVal \* 0.332 + BVal \* 0.500 + 128);

Color\_V = (unsigned char)(RVal \* 0.5 - GVal \* 0.419 - BVal \* 0.0813 + 128);

YUV\_ColorTable[i\*interpolateSize+j].YUVData = (Color\_V << 24) | ((Color\_Y << 16)) | (Color\_U << 8) | (Color\_Y);

}

}

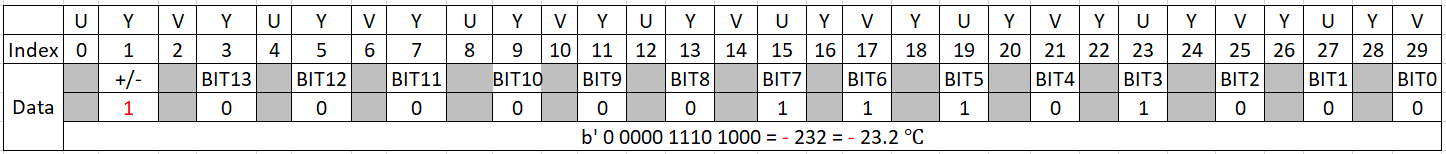
}

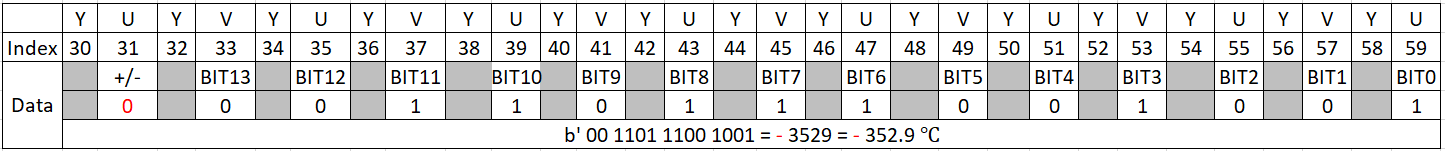
### Data Padding

XCAM contains 2 sensors, which means there are 2 different data to be handled. CIS and thermal image should be formed separately. Default behaviour is to form image on MCU and pass through the video through UVC. Single version output thermal image and Dual version output CIS image with embedded thermal data.

The other way is to get thermal image data by padding bits. The thermal image data is padded into the LSB of Y data for CIS sensor in one data stream, in order to get thermal data to form thermal image at the application side if needed. The linear mapping relationship of thermal data (25.5 °C) and thermal image (RGB) makes calculation can be done bidirectionally. However, sometimes video streaming is not necessary in use case so only data is needed.

Detail is shown below:



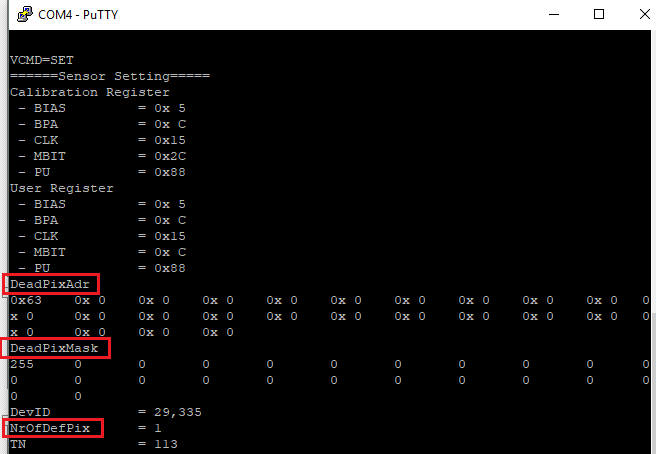


**Each temperature value is sent as 15bits/pixel, first bit represents positive/negative and the following 14 bits for temperature value.**

1. Get YUV data streaming from CIS camera on application side
2. Extract thermal image data from YUV data stream.
3. Convert the temperature values into RGB pixels.
4. Interpolates the RGB pixels so that can be fitted into desired size, i.e.: same as CIS image (640\*480) for overlaying.
5. Please aware of the frame rate difference from CIS and XCAM. CIS image is in higher framerate than XCAM, i.e.: 15fps:6-8 fps

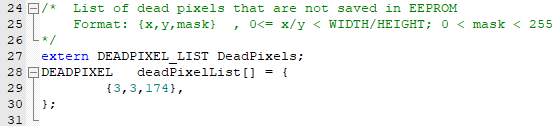
## Dead Pixels Masking

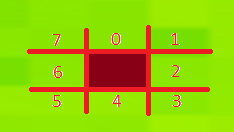
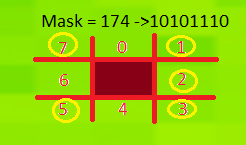
Ideally all the dead pixels are stored in EEPROM NrOfDefPix, DeadPixAdr and DeadPixMask. However, sometimes the information is missing for many reasons.



MI testing application is a good way to scan out any dead pixels that are not being fixed.

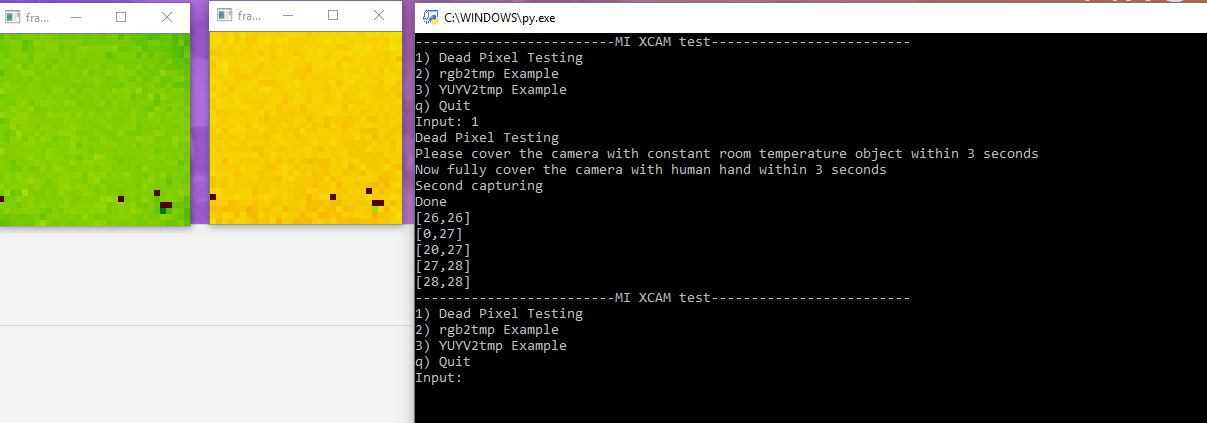
In firmware main.c, deadPixelList[] is the array list saving additional dead pixels. There are 3 data fields in DEADPIXEL, {x,y,mask}. It contains the information of how to do masking the dead pixel. pixel.mask is the integer in decimal based that will be converted into 8-bits binary. The bits are positioning representation of masking on targeted pixel. LSB start from 12 o’clock in clockwise, 0 means ignore. Then pixel averaging is applied to targeted pixel.

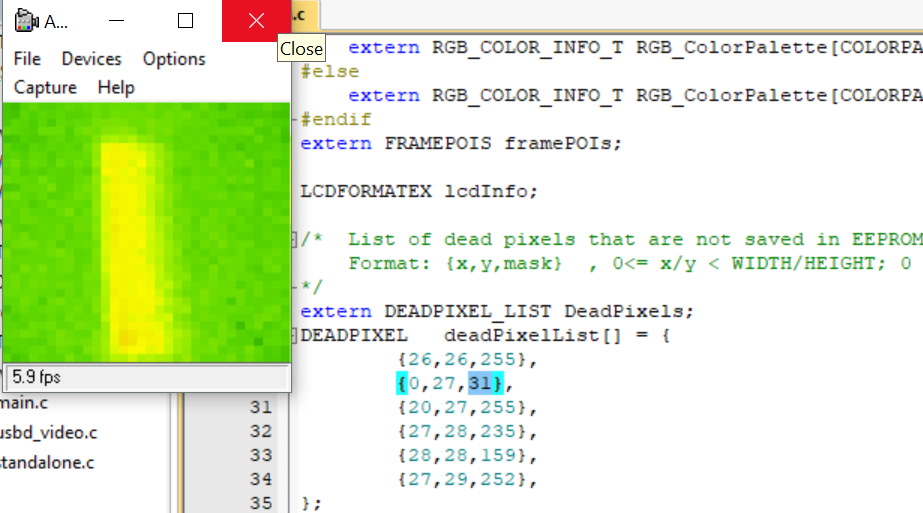


Example

1. Find dead pixels that are not saving in EEPROM
2. Fill in deadPixelList[] and update firmware





## UART/HUART Interface

### Settings

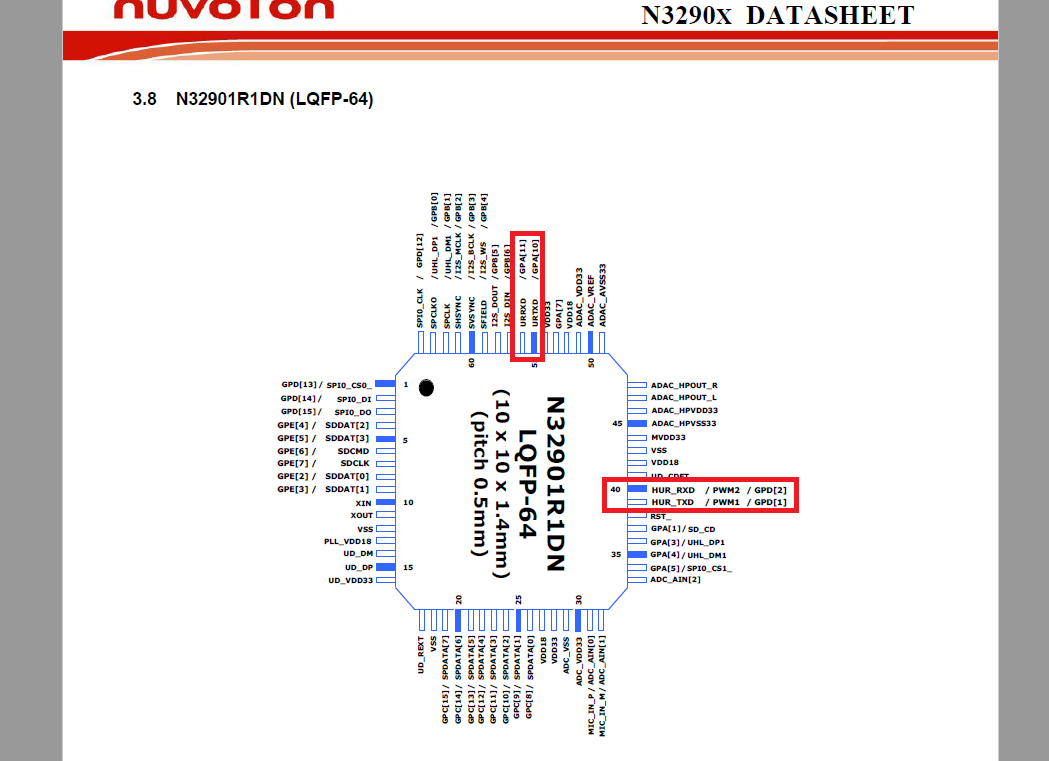
main.c: main() 🡪 N329\_Interface\_init(UART/HUART);

Default Interface: HUART

Baud rate: 115200

Data: 8bit Parity: none

Stop: 1bit Flow control: none

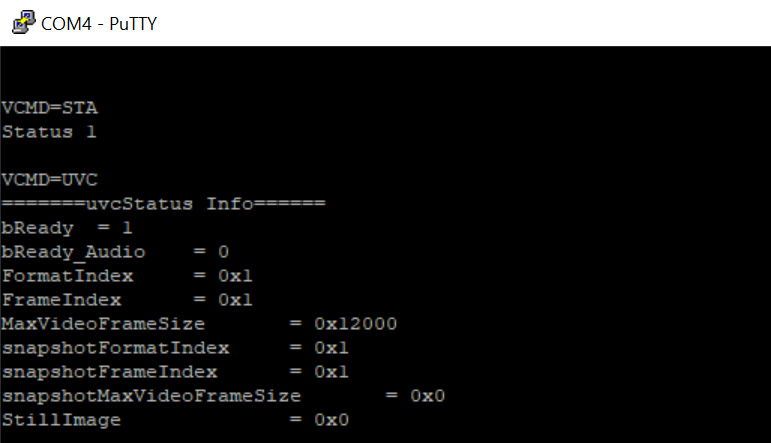


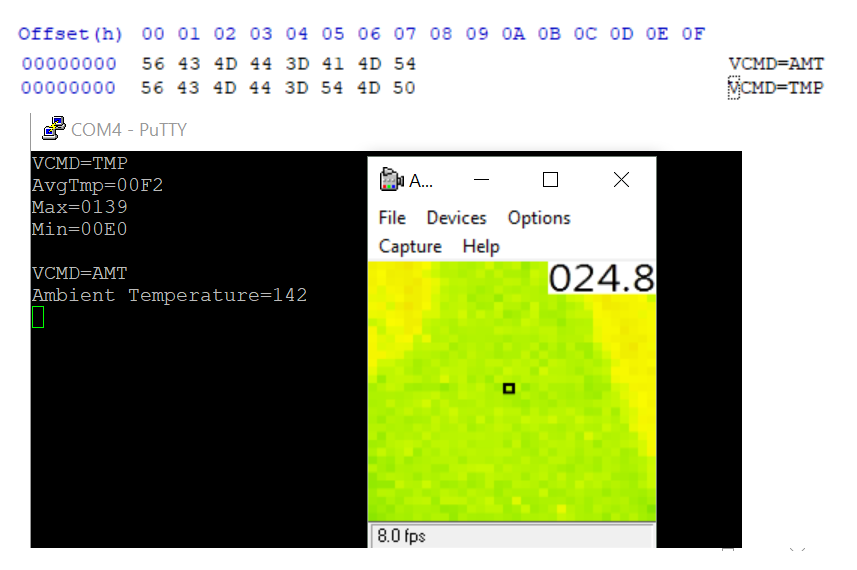
### Output commands

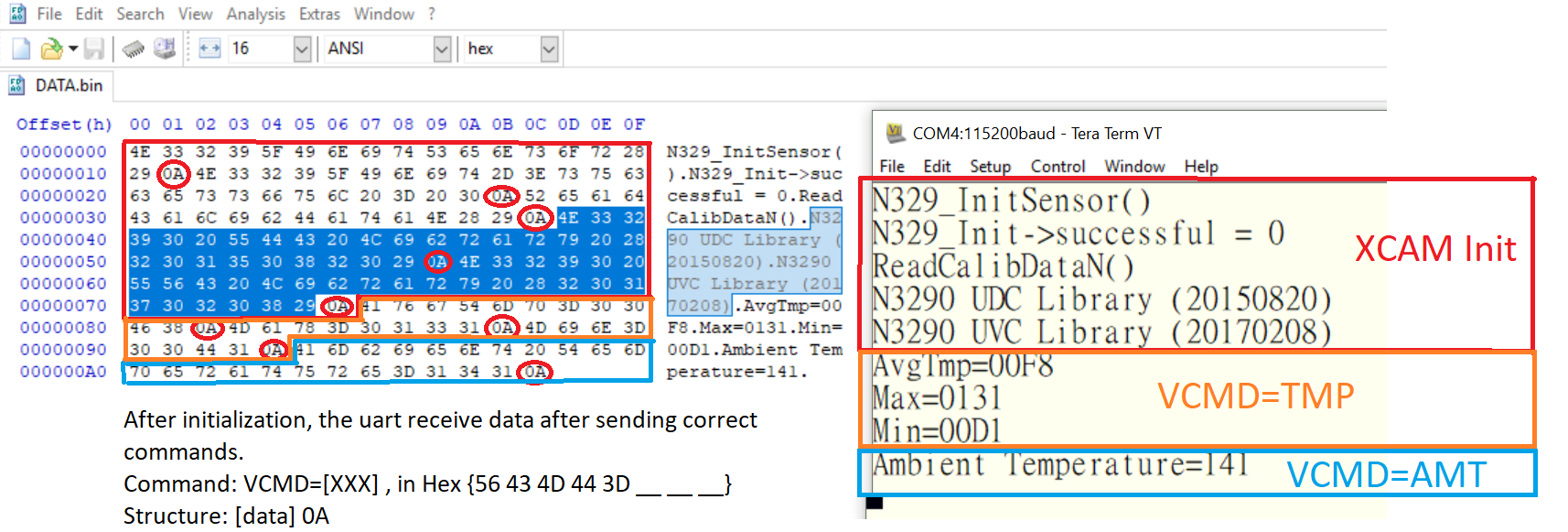
Command: VCMD=$COMMAND {In Hex: 0x56434D443D\_\_\_\_\_\_}

Acknowledgement: data [0A]

|  |  |  |
| --- | --- | --- |
| AMT (Ambient Temperature) | DAT (Temperature Array) | RES (Resolution) |
| SET (Sensor setting) | STA (Sensor Status) | TMP (Display Temperature) |
| UVC (UVC status) | VER (Firmware Version) |  |







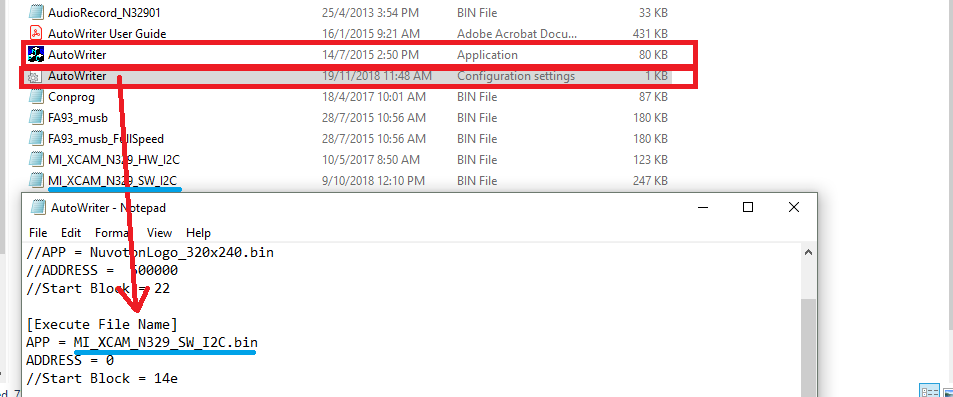
# Build and Test

1. Build main project successfully
2. Download binary file(.bin) to MCU
3. Run the application that support UVC (e.g.: AMCAP.exe)

# Update Firmware

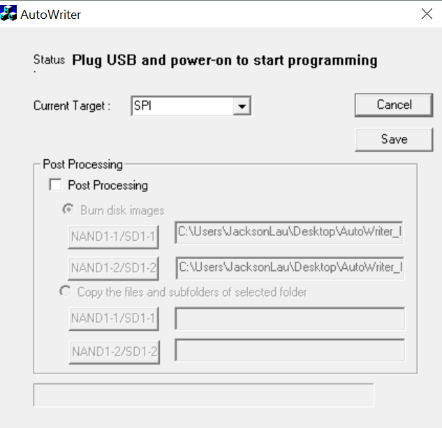
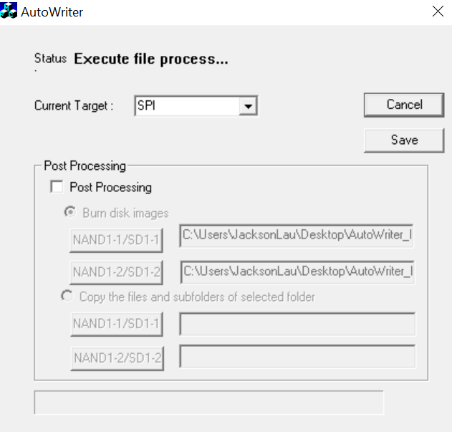
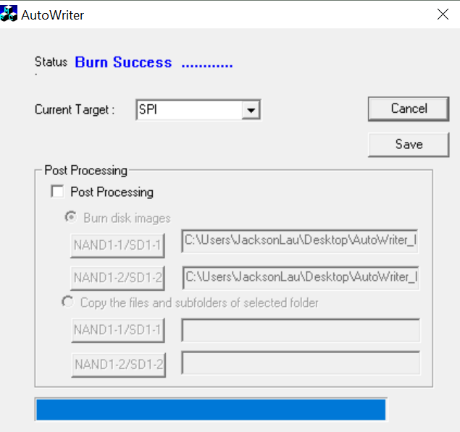
For M480- series, update firmware can be done on Keil by flashing the memory. For N329- series, external software called AutoWriter is needed, to flashing memory for the MCU. Full details can be found on “AutoWriter User Guide.pdf” provided by Nuvoton.

Default binary file generated is called (“MI\_XCAM\_N329\_SW\_I2C.bin”). After building, put this target file into AutoWriter folder at the same directory level of AutoWriter.exe. Open AutoWriter.ini and check if the APP element is pointing to the file name same as target file under [Execute File Name] tag.



Open AutoWrite.exe, set Current Target to SPI. Then power up the XCAM AFTER shorting SW1 pad on the PCB in order to run flashing mode, binary file will be loaded automatically. After seeing the successful message, remember to reopen SW1 pads.

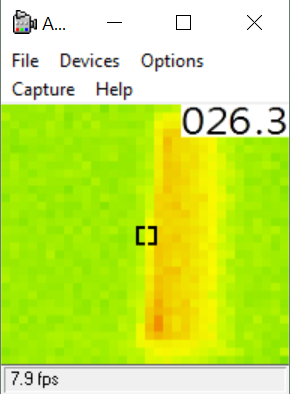
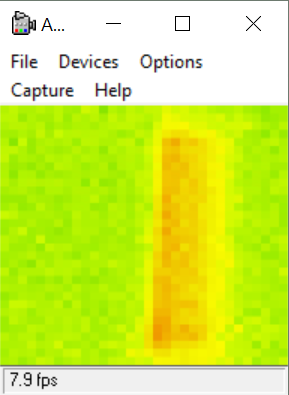


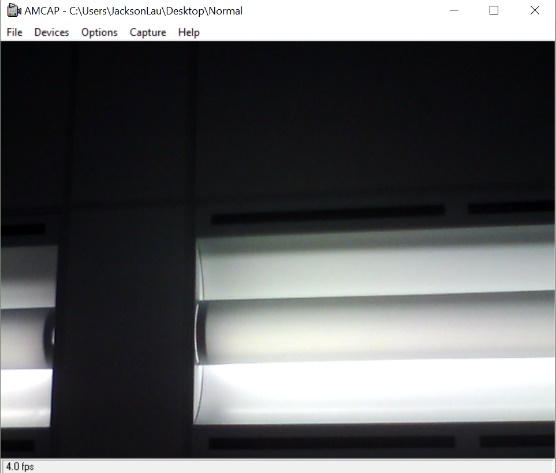
If there are any failure messages, reopen the software and start over.

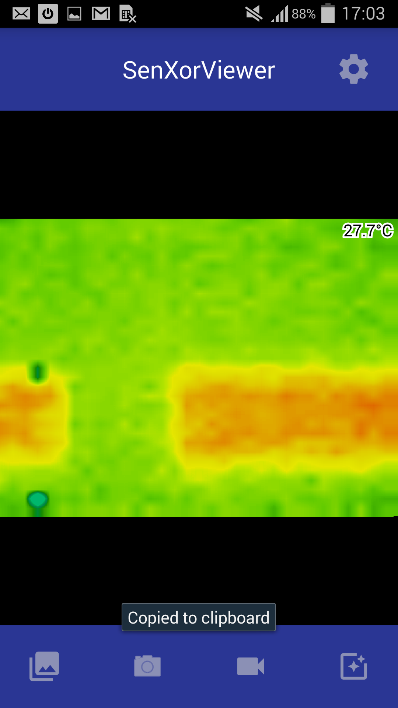
# Demonstration

### Single camera module

### Dual camera module



# Resources

Python based testing application:

<https://github.com/MeridianInno/MI_XCAM_Test>

XCAM Firmware SDK: <https://github.com/MeridianInnovation/MI_XCAM_formalRelease>