

XCAM N329- SDK Manual

|  |  |
| --- | --- |
| Version: | 1.3 |
| Date: | 29th May 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 | Dual camera support + Adaptive colour palette |
| 1.3 | 14 May 2018 | Colour palette update |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

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** 6

**ThermalSensorAPI\_N329.lib** 7

**XCAM Constant** 7

**Customised data structure** 8

**API** 11

Temperature Color Mapping 17

UART/HUART Interface 18

Setting 18

Output commands 18

Build and Test 19

Hybrid images overlaying 19

Procedures 19

Demonstration 20

Single camera module 20

Dual camera module (mobile app) 20

# 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 V3.04 180514**

## Project Setting

### **Header**

"ThermalSensor.h”

“w55fa93\_vpost.h”

“videoclass\_HTPA32.h”

### **Software dependencies**

..\..\..\lib

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

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

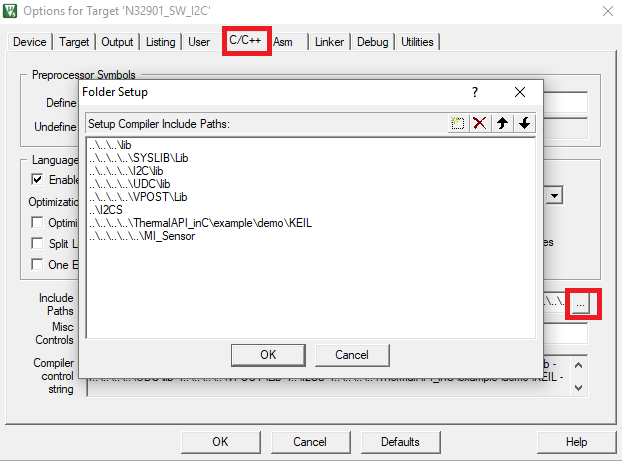
..\..\..\..\UDC\lib

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

..\I2CS

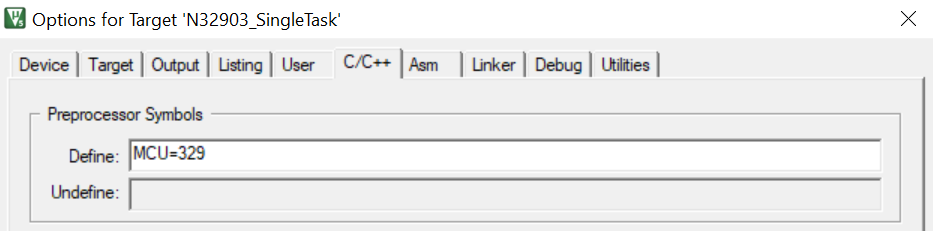
..\..\..\..\ThermalAPI\_inC\example\demo\KEIL

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

****

### **Preprocessor Symbols**

Define: MCU=329



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

#define DEADPIXELCOMPENSATE

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

Prerequisite: EEPROM should have the information of

1) Number of dead pixel and

2) Coordinates of dead pixels

\*If there are any dead pixels next to each other, sensor should be rejected since we are using averaging masking

\*Pre-set from ThermalSensorAPI\_N329.lib

#define POI

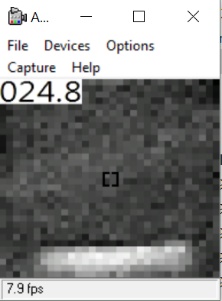
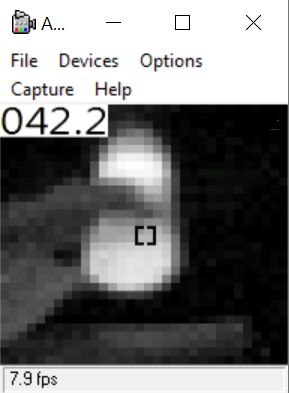
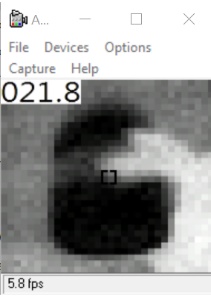
Usage: Point of interests (POIs) recording for each frame. If not define, then FRAMEPOIS object will be empty.

\*Pre-set from ThermalSensorAPI\_N329.lib

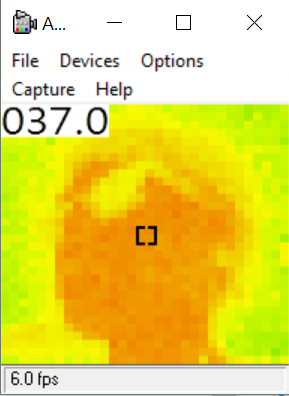
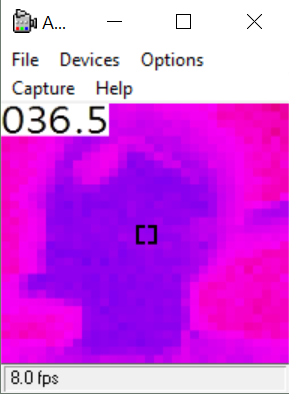
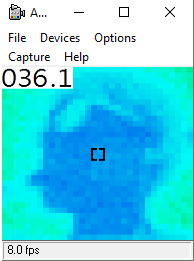
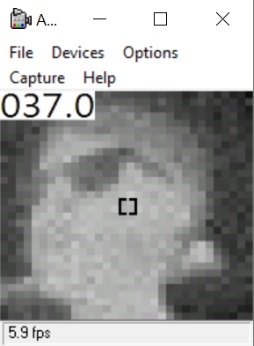
#define COLOR\_ADAPTIVE

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

Better to use monotonic RGB colour table under adaptive mode, i.e.: COLORPALETTE\_BW\_ADAPTIVE

#define COLORPALETTE0/1/2/3/\_BW\_STEP10/\_BW\_STEP25/ COLORPALETTE0\_WIDERANGE

Usage: Define colour palette mode

### **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 |

### **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; |

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

### **API**

**ThermalSensor.h**

void N329\_InitSensor(void);

void N329\_OpenSensor(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[]);

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\_UartDataValid\_Handler (void); |

Handle the UART/HUART command list.

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

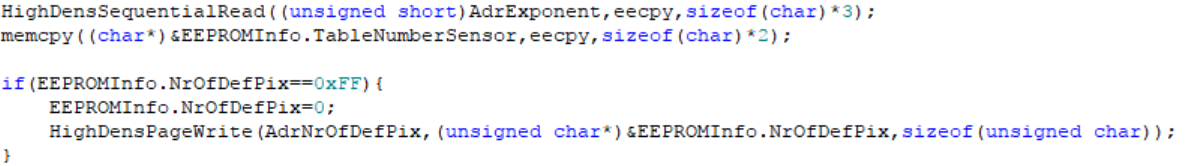


Setup the EEPROM & Sensor address to the device.

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:

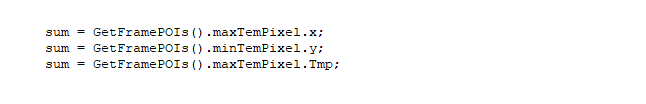
**Y = R \* .299 + G \* .587 + B \* .114;  
U = R \* -.169 + G \* -.332 + B \* .500 + 128.;  
V = R \* .500 + G \* -.419 + B \* -.0813 + 128.;**

Dependencies: RGB\_ColorPalette[] = chosen color palette

YUV\_ColorTable = converted YUV color table from RGB

Return: /

|  |  |
| --- | --- |
| 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); |

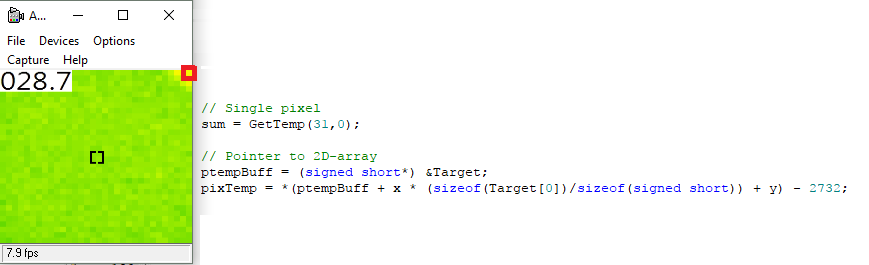


GetTargetPixelIndex() set which pixel temperature should be shown.

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); |

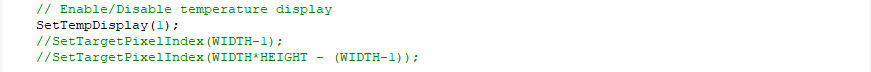
SetTargetPixelIndex() set which pixel we want to show its temperature on the display.

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 Color Mapping

Based on the chosen RGB color palette, it will first converted to the YUV color table with self-defined number of elements in *Create\_color\_table()*. Then, it can map the temperature to its corresponding YUV value by calculating its index.

For adaptive palette, , where 1199 is the index range of the table start from element 0, it is varied depends on the color tale size;

For wide range, the Index is equal to the temperature value;

For other palettes, the Index is equal to the temperature value plus offset.

Example:

We have a BW adaptive palette consist of 61 RGB color elements as follows:

RGB\_COLOR\_INFO\_T RGB\_ColorPalette[ColorPaletteSize]={

{0,0,0},

{4,4,4},

{8,8,8},

{12,12,12},

{16,16,16},

{20,20,20},

{24,24,24},

{28,28,28},

{32,32,32},

{36,36,36},

{40,40,40},

{44,44,44},

{48,48,48},

{52,52,52},

{56,56,56},

{60,60,60},

{64,64,64},

{68,68,68},

{72,72,72},

{76,76,76},

{80,80,80},

{84,84,84},

{88,88,88},

{92,92,92},

{96,96,96},

{100,100,100},

{104,104,104},

{108,108,108},

{112,112,112},

{116,116,116},

{120,120,120},

{124,124,124},

{128,128,128},

{132,132,132},

{136,136,136},

{140,140,140},

{144,144,144},

{148,148,148},

{152,152,152},

{156,156,156},

{160,160,160},

{164,164,164},

{168,168,168},

{172,172,172},

{176,176,176},

{180,180,180},

{184,184,184},

{188,188,188},

{192,192,192},

{196,196,196},

{200,200,200},

{204,204,204},

{208,208,208},

{212,212,212},

{216,216,216},

{220,220,220},

{224,224,224},

{228,228,228},

{232,232,232},

{236,236,236},

{240,240,240},

};

The RGB colour palette is converted to the YUV color table with 1200 elements.

If maxT = 20°C, minT = 30°C, and temp of the point of interest = 25°C, . Thus, the 599th color code on the YUV color table shown on the screen to represent 25°C.

## UART/HUART Interface

### Setting

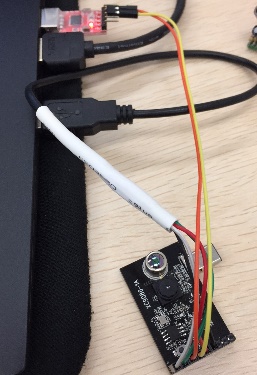
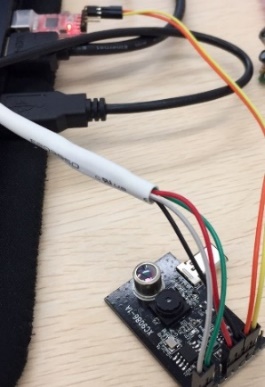
Main.c: main() 🡪 N329\_Interface\_init(UART/HUART);

Baud rate: 115200

Data: 8bit Parity: none

Stop: 1bit Flow control: none

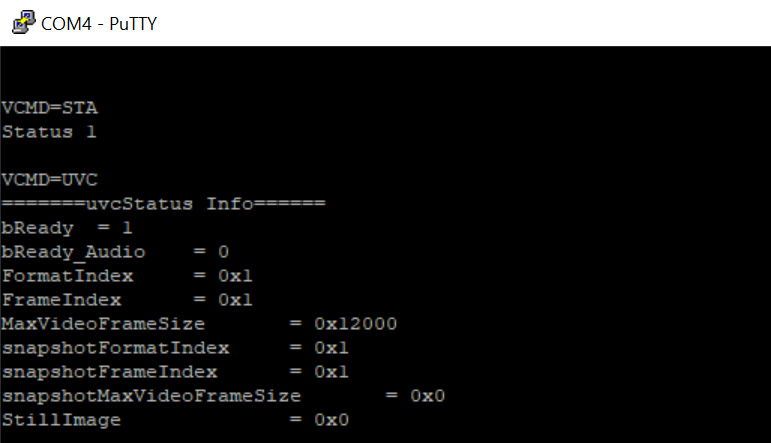
UART HUART

### Output commands

VCMD=$COMMAND

|  |  |  |
| --- | --- | --- |
| 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)

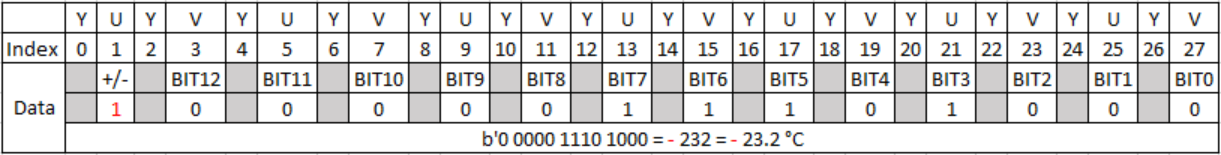
\*Dual camera project will only show CIS image through UVC

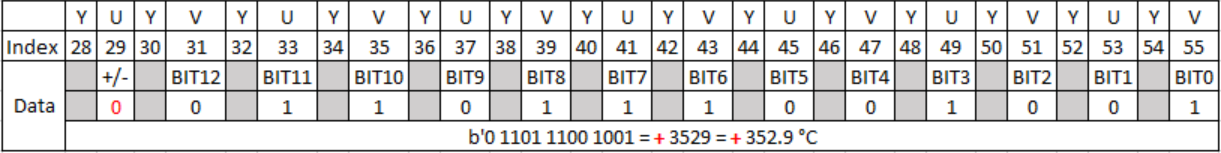
# Hybrid images overlaying

For dual camera module, the data streaming is different from single module. There are two different data to be handled: Thermal image and CIS image.

The thermal image data is padded into the LSB of UV data for CIS sensor in one data stream, normal UVC software will only show the CIS image streaming.

Detail is shown below:





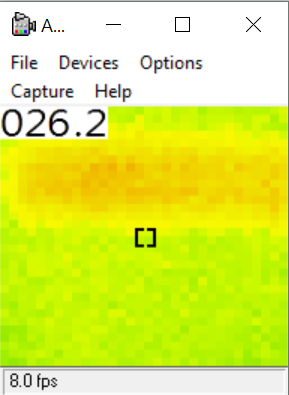
**Each temperature value is sent as 14bits/pixel, first bit represents positive/negative and the following 13 bits for temperature value.**

### Procedures

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

# Demonstration

### Single camera module



### Dual camera module (mobile app)