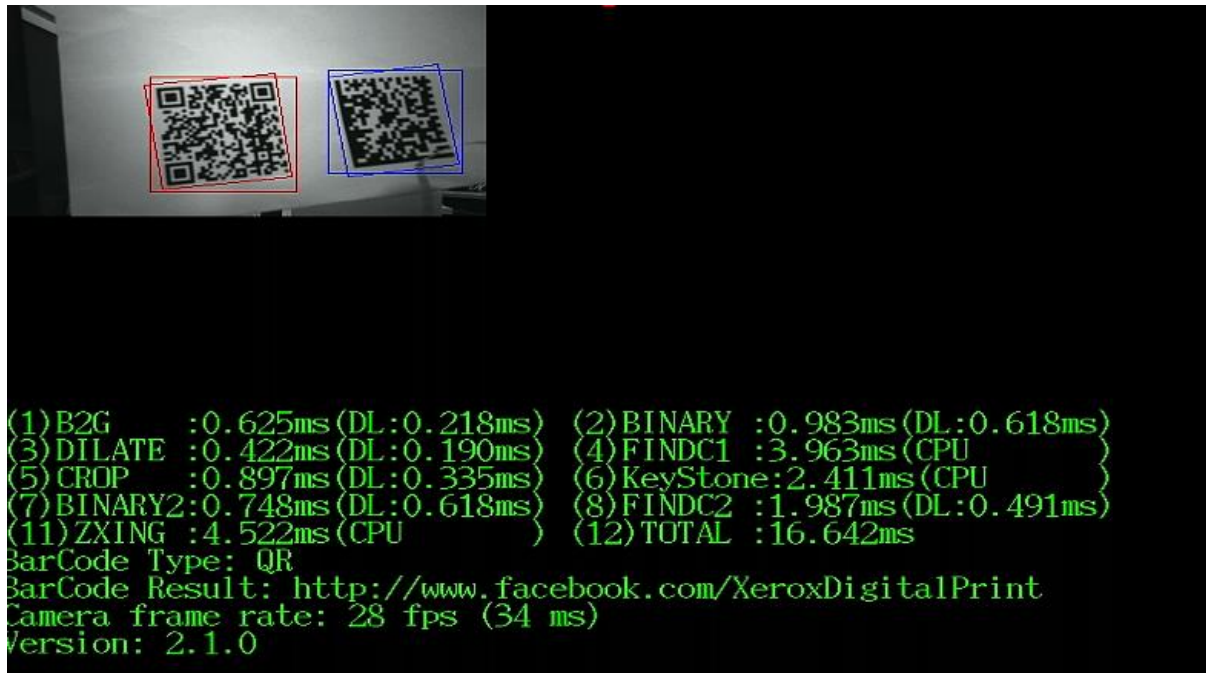


RZ/A2M

Barcode Type Detection Application Note



Introduction

This document describes the RZ/A2M Barcode type detection demo software.

This demo can quickly identify 5 kinds of barcode encoding formats including QR, Micro QR, DataMatrix, AZtec and HanXin code in 8~10ms when data is received from a 1280x720 (1 million) camera.

This demo also integrated ZXing as backend of barcode decoding. When barcode type detection module successfully detected a barcode image, it will output barcode type and cropped barcode image to ZXing module. It will reduce ZXing decoding time. In this sample, it can support 3 kinds of barcode decoding: QR, DataMatrix and AZtec

Target Device

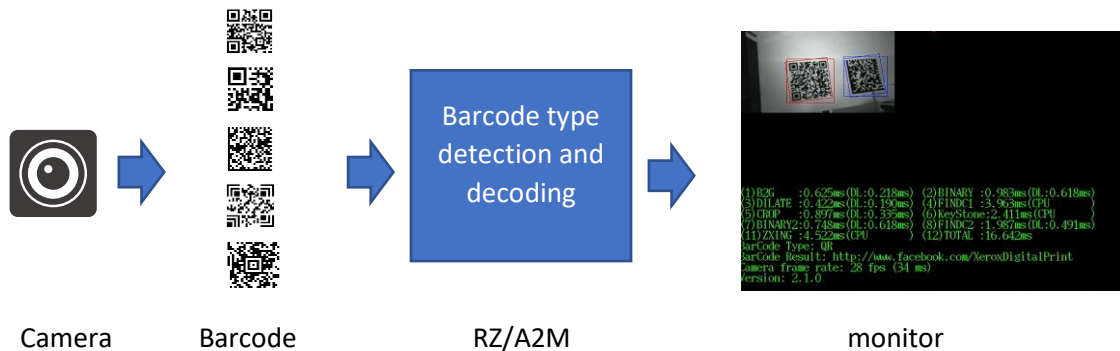
RZ/A2M

Contents

1. Overview	3
2. Operation Confirmation Conditions.....	5
2.1. Evaluation Board Kit setup.....	5
2.1.1. Evaluation Board Kit setup (quick start)	5
2.1.2. Raspberry PI camera holder (3D model)	6
2.2. GR-MANGO board setup.....	8
2.2.1. Raspberry PI camera holder (3D model)	8
3. Software.....	9
3.1. Folder Structure	9
3.2. Display type definition	10
3.3. DRP libraries.....	11
3.4. External RAM	11
4. Image Pre-Processing.....	11
4.1. The overview of barcode type detection and decoding	11
4.2. Pre-Processing step “Bayer2gray_thinning”	12
4.3. Pre-Processing step “Binarization_Adaptive”	14
4.4. Pre-Processing step “Dilate”	15
4.5. Pre-Processing step “Find Contour” (OpenCV)	17
4.6. Pre-Processing step “Bayer2Gray_Cropping”	17
4.7. Pre-Processing step “Keystone”	19
4.8. Pre-Processing step “Binarization_Adaptive”	19
4.9. Pre-Processing step “Find Contour” (DRP)	20
4.10. Pre-Processing step “Barcode Symbol Detection”	21
5. Barcode Decoding.....	23
6. Parameter Tuning	23
7. Pre-compiled binary Downloading	24

1. Overview

The demo detects different barcode format taken by a camera. The result is shown on a display (HDMI monitor).



The Barcode type detection demo software is available for 2 different platforms, please refer to below link for general information about the EVK and GR-MANGO platform:

1. RZ/A2M Evaluation Board Kit
<https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/rz-cortex-a-mpus/rza2m-evaluation-kit-rza2m-evaluation-kit>
2. Gadget Renesas board "GR-MANGO"
<https://www.renesas.com/eu/en/products/gadget-renesas/boards/gr-mango>

Used e²studio version: e²studio 2021.04 (2021-04 (21.4.0))

In this sample, we provided 2 different e2 studio projects accordingly.

e²studio project name: "rza2m_barcode_type_freertos_gcc_evk"

e²studio project name: "rza2m_barcode_type_freertos_gcc_grmango"

There are a few platform-dependant differences. These are marked with the e²studio project name.

The user switches are used to select different options:

	rza2m_barcode_type_freertos_gcc_evk	rza2m_barcode_type_freertos_gcc_grmango
select the image and information shown at the display (*1)	SW3	SW2
reset	n.a.	SW1

(*1) Three different camera image display modes

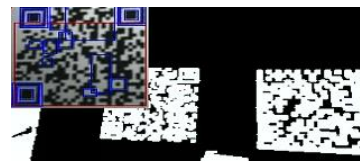
1. Grayscale Image
2. Binarization Image
3. Binarization Image + Cropped & Rotated grayscale barcode image



Mode1(Default)



Mode2



Mode3

The demo software is based on the “2D Barcode package” (r01an4487xx0109-rza2m-2dbarcode-swpkg-gcc).

The “2D Barcode package” has been released on Renesas website. It is available for the RZ/A2M Evaluation Board Kit (EBK) and GR-Mango board.

The latest version of the “2D Barcode package” is available for download at

<https://www2.renesas.cn/cn/zh/document/scd/rza2m-group-rza2m-2d-barcode-package-v109-sample-code>

<https://www2.renesas.cn/cn/zh/document/scd/rza2m-group-rza2m-2d-barcode-package-gr-mango-v100-sample-code>

2. Operation Confirmation Conditions

2.1. Evaluation Board Kit setup

Please setup the development kit board as described in
rza2m_barcode_type_freertos_gcc_evk/doc/readme-e.txt
Especially the setting for DIP switches and jumpers needs to be followed.

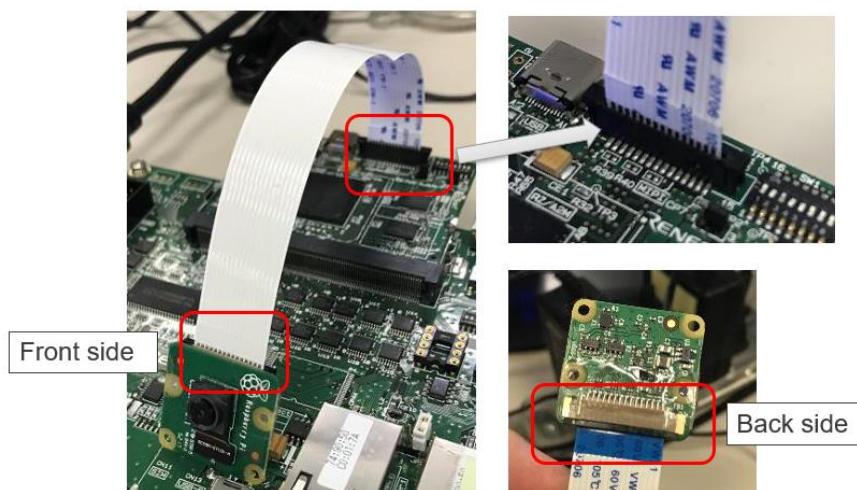
Additional information can be found in the:

rza2m_barcode_type_freertos_gcc_evk/doc/r01qs0027ej0108-rza2m-quick-guide-gcc.pdf

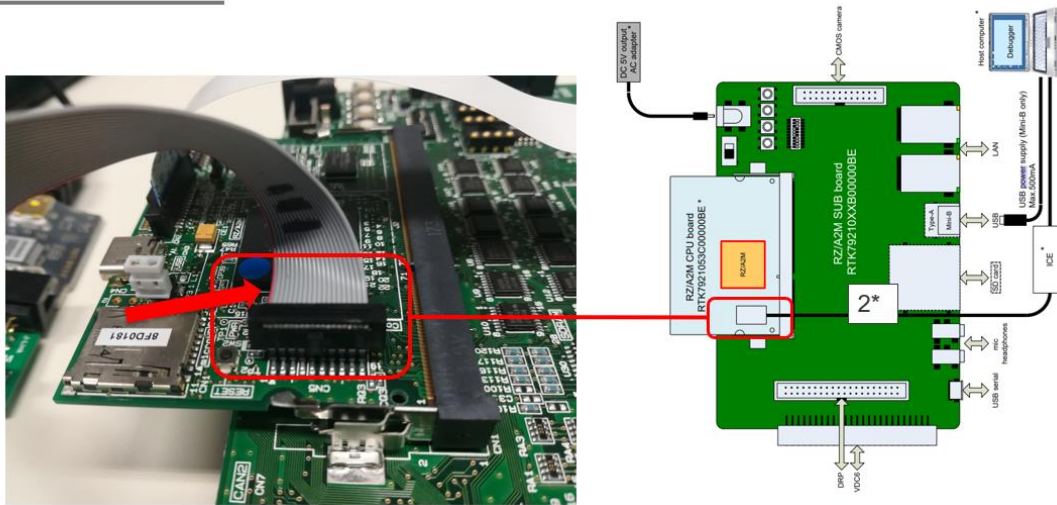
rza2m_barcode_type_freertos_gcc_evk/doc/r20ut4535ej0102-e2studio.pdf

2.1.1. Evaluation Board Kit setup (quick start)

MIPI CAMERA CONNECTION

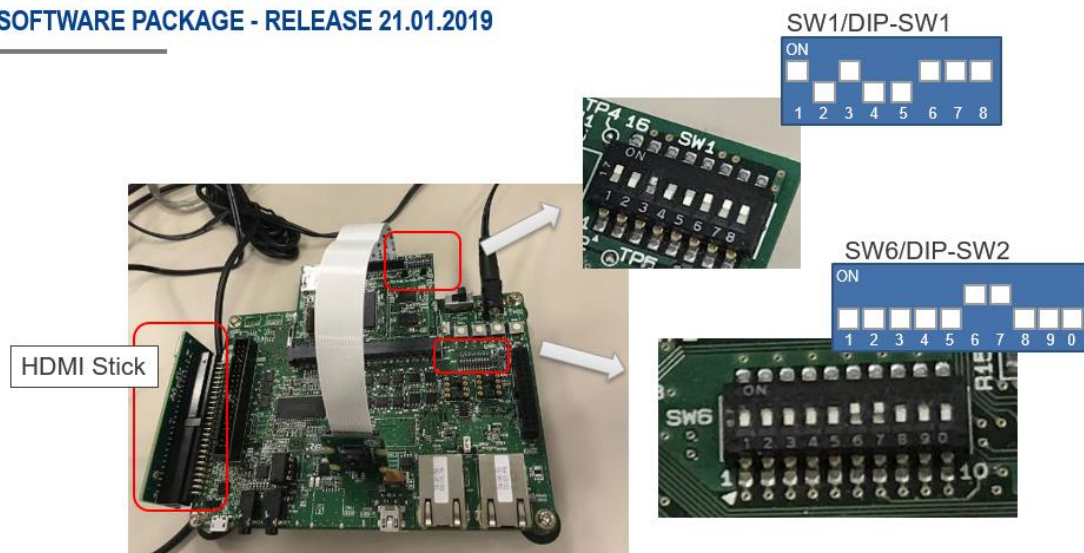


J_LINK CONNECTION



DIP SWITCHES AND JUMPERS

RZ/A2M SOFTWARE PACKAGE - RELEASE 21.01.2019

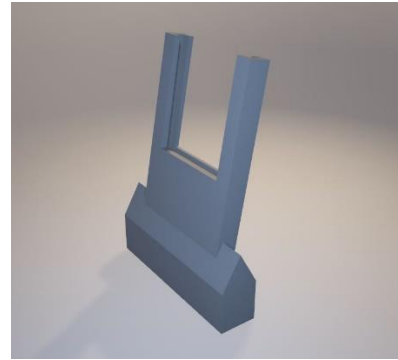


2.1.2. Raspberry PI camera holder (3D model)

The following camera holder is quite useful to stabilize the camera.

RZ/A2M development kit gadget: Raspberry PI camera holder (3D model)

https://renesasrulz.com/rz/m/rz_files/3399



2.2. GR-MANGO board setup

Please setup the development kit board as described in
`rza2m_barcode_type_freertos_gcc_grmango/doc/readme-e.txt`

“(4) Downloading sample code

Connect GR-MANGO CN1 and PC via USB cable.

PC detects the GR-MANGO as MBED drive.

Drag-and-drop the binary file

(e.g. `rza2m_barcode_type_freertos_gcc_grmango/HardwareDebug/
rza2m_barcode_type_freertos_gcc_grmango.bin`)
to the MBED drive.

(5) Executing sample code

Connect the camera to GR-MANGO CN13 and connect display to CN9 via HDMI cable.

Ensure that complete to download program, and push reset button on GR-PEACH.”

Additional information can be found in the:

`rza2m_barcode_type_freertos_gcc_grmango/doc/r01an5595ej0200-rza2m-swpkg-grmango-gcc.pdf`

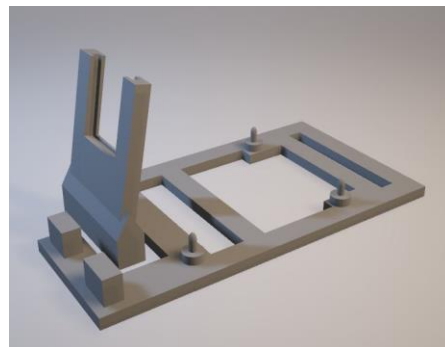
`rza2m_barcode_type_freertos_gcc_grmango/doc/r01qs0042ej0101-rza2m-quick-guide-grmango-gcc.pdf`

2.2.1. Raspberry PI camera holder (3D model)

The following camera holder is quite useful to stabilize the camera.

RZ/A2M GR-MANGO board gadget: board mount with Raspberry PI camera holder (3D model)

https://renesasrulz.com/rz/m/rz_files/3402



3. Software

3.1. Folder Structure

```

rza2m_barcode_type_freertos_gcc_evk or
rza2m_barcode_type_freertos_gcc_grmango
+---bootloader
+---doc
+---FlashTools
+---generate
|   +---compiler
|   +---configuration
|   +---drivers
|   +---gr_mango_boot          (RZA2M_GR_MANGO only)
|   +---os_abstraction
|   +---sc_drivers
|   |   +---r_cbuffer
|   |   +---r_ceu
|   |   +---r_drp
|   |   |   +---doc
|   |   |   +---drp_custom
|   |   |   +---drp_lib
|   |   |   +---inc
|   |   |   \---src
|   |   +---r_mipi
|   |   +---r_octabus          (RZA2M_GR_MANGO only)
|   |   +---r_ostm
|   |   +---r_riic
|   |   +---r_rvapi
|   |   +---r_scifa
|   |   \---r_vdc
|   \---system
\---src
    +---config_files
    +---FreeRTOS
    +---renesas
    |   +---application
    |   |   +---common
    |   |   |   +---camera
    |   |   |   +---console
    |   |   |   +---perform
    |   |   |   +---port_settings
    |   |   |   \---render
    |   |   +---inc
    |   |   |   +---camera
    |   |   |   \---lcd
    |   |   \---hwsetup
    |   |       +---r_octabus_middleware    (RZA2M_GR_MANGO only)
    +---user_prog
    \---zxing_lib

opencv_neon
+---3rdparty
+---calb3d
... ..
+---Release          (pre-compiled openCV lib)
\---videostab

```

FreeRTOS is open-source software distributed under the MIT license.

Regarding the MIT license, please refer to

<https://opensource.org/licenses/mit-license.php>

FreeRTOS is a real-time operation system kernel for embedded microcomputer.

This sample program is based on FreeRTOS OS Abstraction Version 3.5

openCV is open-source software distributed under 3-clause BSD License.

Regarding the 3-clause BSD License, please refer to

<https://github.com/opencv/opencv/blob/4.4.0/LICENSE>

openCV is an open source computer vision library.

This sample program is based on openCV 3.2.0

3.2. Display type definition

The RZ/A2M Evaluation Board Kit offers different options to connect a display.

To change the display type, please open header file

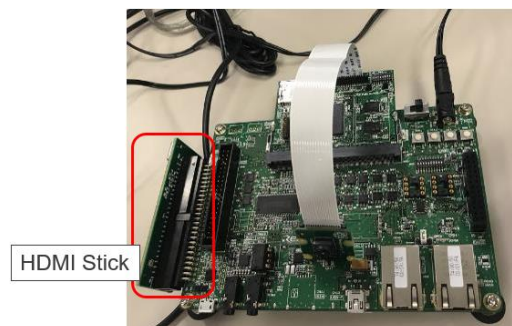
```
rza2m_barcode_type_freertos_gcc_evk/src/renesas/application/inc/lcd_panel.h
```

and map LCD_PANEL with one of the pre-defined options (e.g. LCD_PANEL_RSK or LCD_PANEL_DVI).

LCD_PANEL_RSK is the label for a Renesas touch display that can be connected to the Evaluation board kit directly.



LCD_PANEL_DVI is the label for HDMI displays that can be connected via the HDMI interface stick that needs to be connected to the development board.



RZ/A2M Evaluation Board Kit and GR-MANGO board:

If LCD_PANEL_DVI (HDMI) is selected, please check header file

```
rza2m_barcode_type_freertos_gcc_evk/src/renesas/application/inc/lcd/pc_monitor.h
```

and modify the define for SELECT_MONITOR to select the most suitable configuration for the connected monitor.

3.3. DRP libraries

The Software uses a mixture of released DRP libraries and modified DRP libraries that have been developed at Renesas Electronics China.

The libraries are given in

```
/rza2m_barcode_type_freertos_gcc_*/generate/sc_drivers/r_drp
/rza2m_barcode_type_freertos_gcc_*/generate/sc_drivers/r_drp/drp_custom
```

Please check the documents in sub-directories

r_drp/doc,
for detailed information.

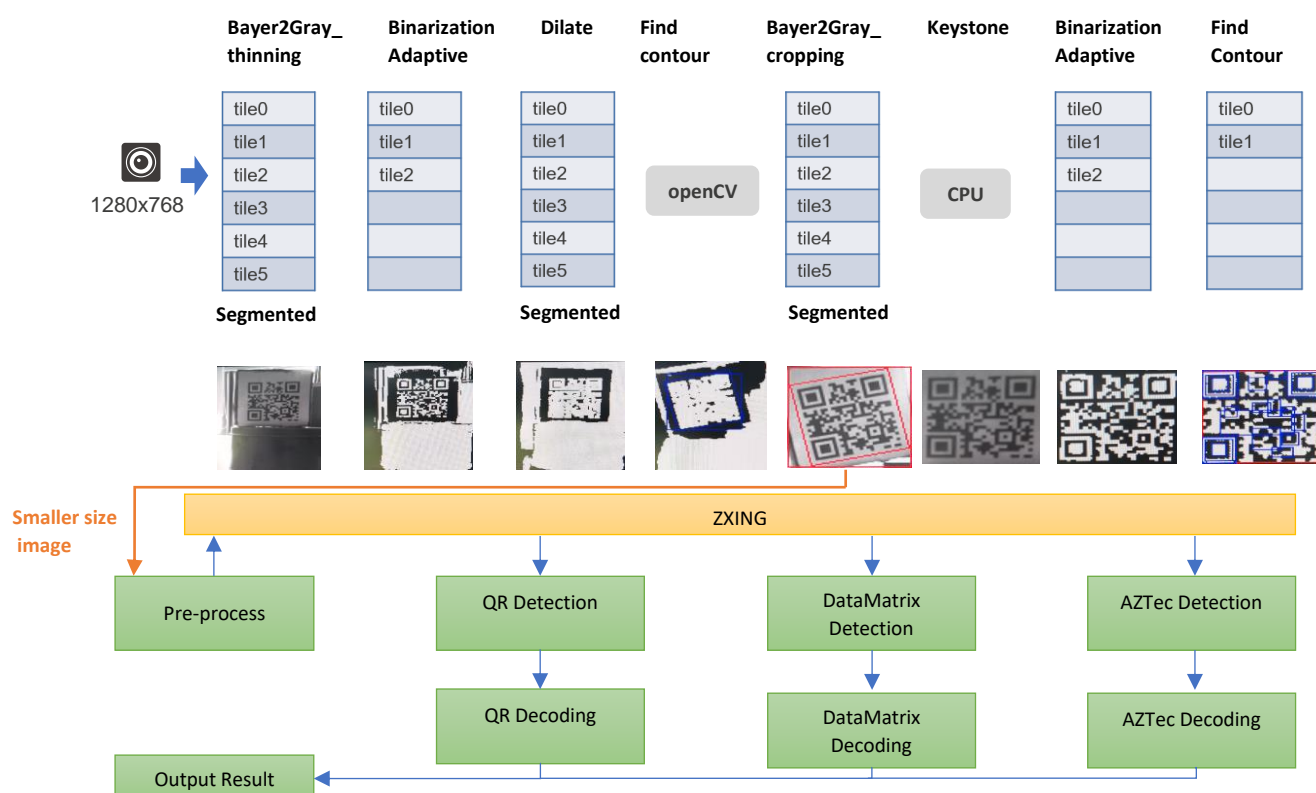
3.4. External RAM

In “rza2m_barcode_type_freertos_gcc_evk” sample, hyperRAM is used for RAM extension to allocate some bss stack buffer and FreeRTOS heap buffer

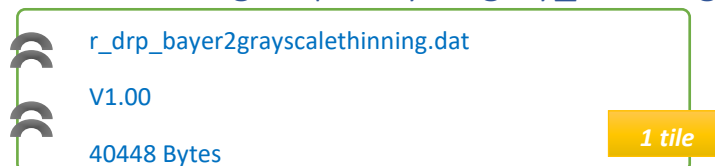
In “rza2m_barcode_type_freertos_gcc_grmango” sample, OctaRAM is used for RAM extension to allocate some bss stack buffer and FreeRTOS heap buffer

4. Image Pre-Processing

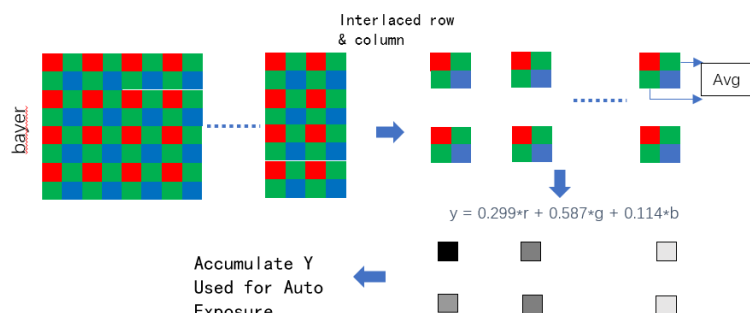
4.1. The overview of barcode type detection and decoding



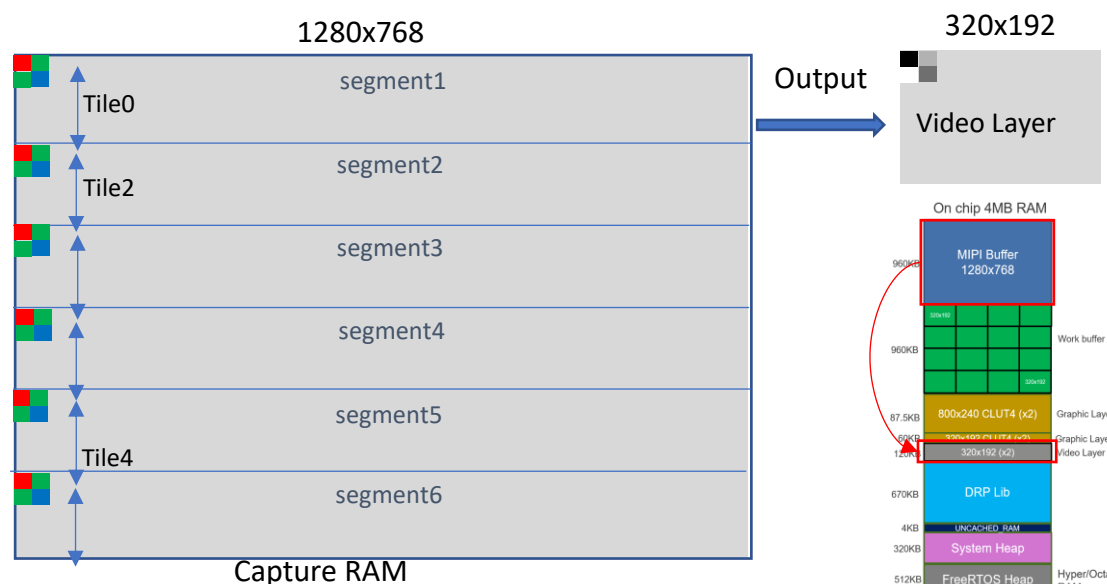
4.2. Pre-Processing step “Bayer2gray_thinning”



In order to speed up the image processing, we first load a Bayer2GrayThinning DRP acceleration library to downscale the sample image resolution to 1/4 of the original size, and then convert the image format from Bayer to grayscale. The brightness level of each frame of image is determined and then fed back to the CMOS sensor to adjust the automatic exposure parameters



Since the Bayer2GrayThinning library only occupies 1 Tile of DRP hardware resources, in this step, the multi-tile parallel processing feature of DRP is used to divide each frame of 1280x768 image into 6 1280x128 regions, which are parallelized by 6 DRP processing units. In this way, the processing speed can be increased by 6 times, and the pixel-level processing inside each processing unit is also parallelized.



Sample Code

```
ret_val = R_DK2_Load(&g_drp_lib_bayer2gray_thinning[0],
    R_DK2_TILE_0 | R_DK2_TILE_1 | R_DK2_TILE_2 |
    R_DK2_TILE_3 | R_DK2_TILE_4 | R_DK2_TILE_5,
    R_DK2_TILE_PATTERN_1_1_1_1_1_1, NULL, &cb_drp_finish,
    &drp_lib_id[0]);
DRP_DRV_ASSERT(ret_val);
```

Load DRP Lib to Tile0~5

6 Tiles execute in parallel

```

for (tile_no = 0; tile_no < R_DK2_TILE_NUM; tile_no++) Configure 6 tiles
{
    /* Set the address of buffer to be read/write by DRP */

    param_b2graythinning[tile_no].src = input_address + (input_width * (input_height / R_DK2_TILE_NUM)) * tile_no; Set input/output address
    param_b2graythinning[tile_no].dst = output_address + (input_width/DRP_RESIZE * (input_height/DRP_RESIZE / R_DK2_TILE_NUM)) * tile_no;
    R_MMU_VAtoPA((uint32_t*)&ave_result[tile_no], (uint32_t*)&param_b2graythinning[tile_no].accumulate);

    /* Set Image size */ Set brightness output address, it is declared in uncached RAM

    param_b2graythinning[tile_no].width = (uint16_t)input_width; The width remains the same and the height is 1/6 of the original height
    param_b2graythinning[tile_no].height = (uint16_t)input_height / R_DK2_TILE_NUM;

    param_b2graythinning[tile_no].area1_offset_x = 0;
    param_b2graythinning[tile_no].area1_offset_y = 0;

    param_b2graythinning[tile_no].area1_width = input_width / DRP_RESIZE; Statistic the brightness of whole frame
    param_b2graythinning[tile_no].area1_height = input_height / DRP_RESIZE / R_DK2_TILE_NUM;

    /* Initialize variables to be used in termination judgment of the DRP application */

    drp_lib_status[tile_no] = DRP_NOT_FINISH; Reset the status of each tile, this value will be updated in DRP interrupt

    /*******/
    /* Start DRP Library */
    /*******/

    ret_val = R_DK2_Start(drp_lib_id[tile_no], (void *)&param_b2graythinning[tile_no], sizeof(r_drp_bayer2gray_thinning_t)); Start DRP
    DRP_DRV_ASSERT(ret_val);
}

```

```

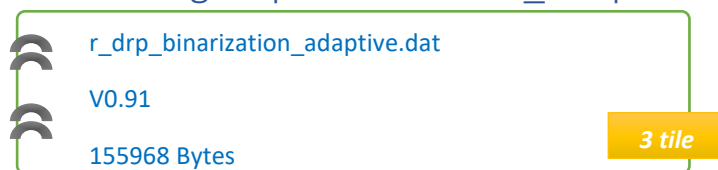
for (tile_no = 0; tile_no < R_DK2_TILE_NUM; tile_no++)
{
    While (drp_lib_status[tile_no] == DRP_NOT_FINISH); Wait for each segment to complete; Total of about 0.4 ms
}
for (tile_no = 1; tile_no < R_DK2_TILE_NUM; tile_no++)
{
    ave_result[0] += ave_result[tile_no]; Accumulate the brightness of each segment
}

```

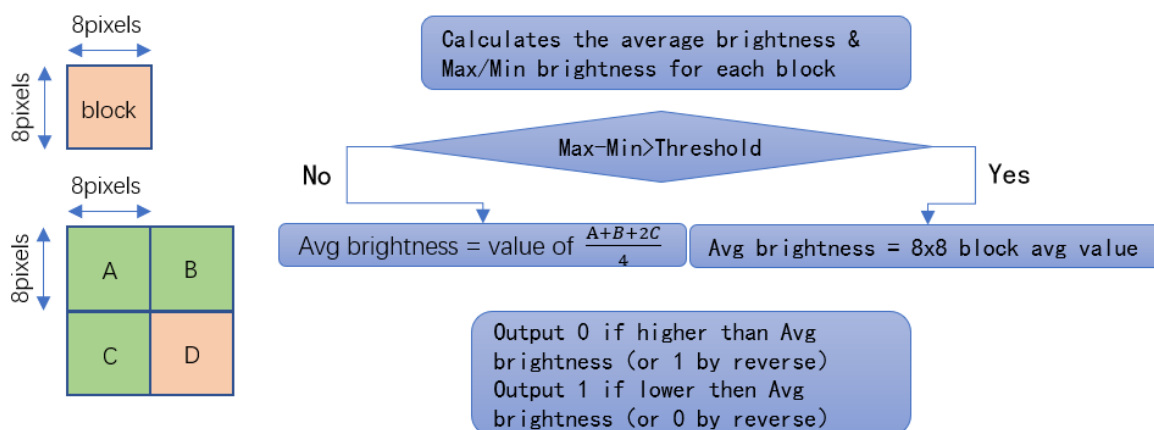
We can complete the processing of one frame of image in 1280x768 resolution in 0.6ms, including 0.2ms DRP library loading time and 0.4ms image data processing time. After processing in this step, we can get a 320x192 grayscale image.



4.3. Pre-Processing step “Binarization_Adaptive”

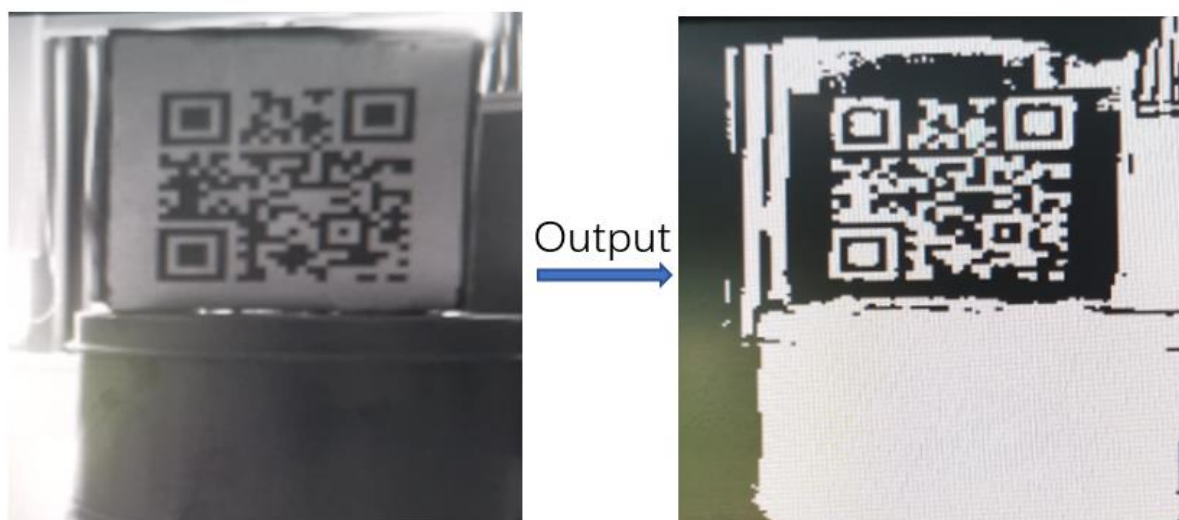


After getting the grayscale image, we load a binarization adaptive DRP library and convert the image into a black and white image in 0x00 and 0xFF. This DRP library can dynamically calculate the binarization threshold based on the average gray value of each 8x8 pixel area, which can effectively solve the problem of how to choose the binarization threshold in scenes with different brightness.



We can set the parameters of the binary conversion and control the output image to be output in reverse mode, that is, the black part of the two-dimensional barcode image is output as white, so that it is convenient to extract the contour information of the white part in the next step.

The total processing time of this step is about 0.9ms, including 0.6ms DRP library loading time and 0.3ms image data processing time.



Sample Code

```
ret_val = R_DK2_Load(&g_drp_lib_binarization[0], R_DK2_TILE_0,
    R_DK2_TILE_PATTERN_3_1_1_1,
    NULL, &cb_drp_finish, &drp_lib_id[0]);
DRP_DRV_ASSERT(ret_val);
ret_val = R_DK2_Activate(drp_lib_id[TILE_0], 0);
```

Load DRP Lib to the position of Tile0.
It will use the resource of Tile0, Tile1 and Tile2

```
param_binarization.src = input_address;
```

```
param_binarization.dst = output_address;
```

Write image to first block of the work buffer

```
param_binarization.work = work_address;
```

Use second block of the work buffer

```
param_binarization.width = (uint16_t)input_width & (~31);
```

Width should be multiple of 32
Height should be multiple of 8

```
param_binarization.height = (uint16_t)input_height & (~7);
```

```
param_binarization.range = range;
```

```
param_binarization.invert = 1;
```

Enable inversus output

```
drp_lib_status[TILE_0] = DRP_NOT_FINISH;
```

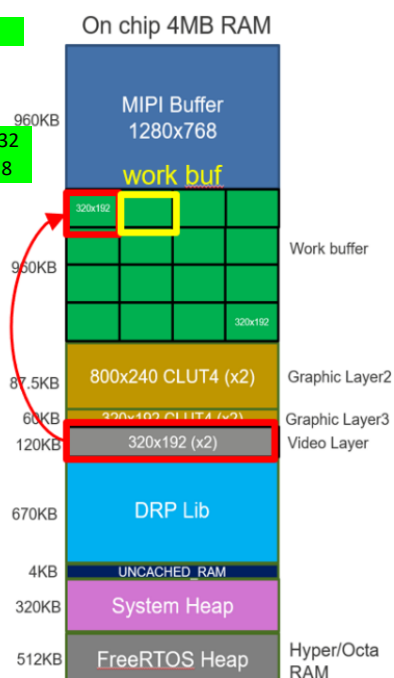
```
ret_val = R_DK2_Start(drp_lib_id[TILE_0], (void *)&param_binarization,
    sizeof(r_drp_binarization_adaptive_t));
```

```
//DRP_DRV_ASSERT(ret_val);
```

```
while (drp_lib_status[TILE_0] == DRP_NOT_FINISH);
```

Wait for DRP finish, ~0.3ms

```
}
```



4.4. Pre-Processing step “Dilate”

r_drp_erode.dat

V1.00

57344 Bytes

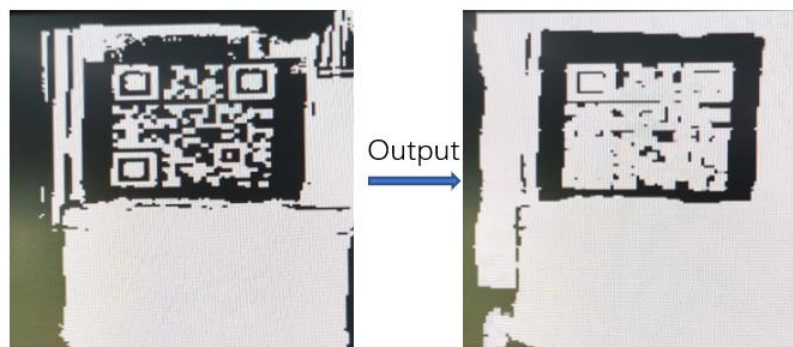
1 tile

After obtaining the binarized image, we load a Dilate DRP library to implement the expansion function. The basic implementation principle is as follows;

15	18	30
5	10	50
7	18	20

- Scan the input image with a 3x3 block with a step size of 1 pixel
- The new data output at the center of the scan window is the maximum value within the range of 3x3
- Realization effect: Expand white area and swallow the enclosed black area
- The number of iterations can be configured, the more times the larger the white extension range

In the expanded image, the area containing the QR code data is basically connected to a large area containing the entire QR code.



The total processing time is about 0.4ms, including 0.2ms DRP library loading time and 0.2ms image data processing time.

Sample Code

```
ret_val = R_DK2_Load(&g_drp_lib_dilate[0], R_DK2_TILE_0 | R_DK2_TILE_1 |
    R_DK2_TILE_2 | R_DK2_TILE_3 |
    R_DK2_TILE_4 | R_DK2_TILE_5,
    R_DK2_TILE_PATTERN_1_1_1_1_1_1, NULL, &cb_drp_finish, &drp_lib_id[0]);
ret_val = R_DK2_Activate(drp_lib_id[TILE_0] | drp_lib_id[TILE_1] |
    drp_lib_id[TILE_2] | drp_lib_id[TILE_3] | drp_lib_id[TILE_4] |
```

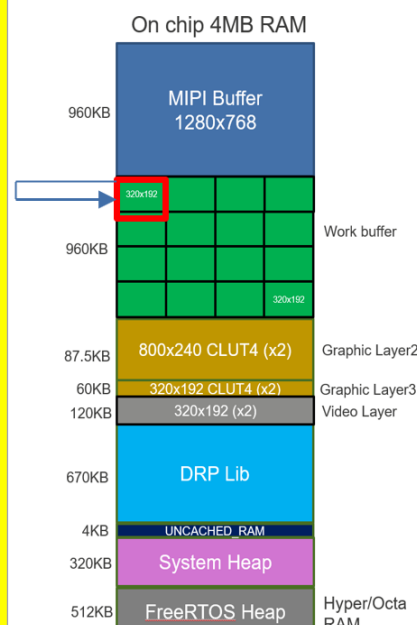
Load DRP Lib to Tile0~5

6 Tiles execute in parallel

```
for (loop = 0; loop < times; loop++){
    for (tile_no = 0; tile_no < R_DK2_TILE_NUM; tile_no++){
        if (loop == 0){
            param_dilate[tile_no].src = input_address + (input_width
                * (input_height / R_DK2_TILE_NUM)) * tile_no;
            param_dilate[tile_no].dst = output_address1 + (input_width
                * (input_height / R_DK2_TILE_NUM)) * tile_no;
        }
        else{
            if (loop % 2 == 1){
                param_dilate[tile_no].src = output_address1 +
                    (input_width * (input_height / R_DK2_TILE_NUM)) * tile_no;
                param_dilate[tile_no].dst = output_address2 +
                    (input_width * (input_height / R_DK2_TILE_NUM)) * tile_no;
            }
            else{
                param_dilate[tile_no].src = output_address2 +
                    (input_width * (input_height / R_DK2_TILE_NUM)) * tile_no;
                param_dilate[tile_no].dst = output_address1 +
                    (input_width * (input_height / R_DK2_TILE_NUM)) * tile_no;
            }
        }
        param_dilate[tile_no].width = (uint16_t)input_width;
        param_dilate[tile_no].height = (uint16_t)input_height /
            R_DK2_TILE_NUM;
        param_dilate[tile_no].top = (tile_no == TILE_0) ? 1 : 0;
        param_dilate[tile_no].bottom = (tile_no == TILE_5) ? 1 : 0;
        drp_lib_status[tile_no] = DRP_NOT_FINISH;
        ret_val = R_DK2_Start(drp_lib_id[tile_no], (void *)
            &param_dilate[tile_no], sizeof(r_drp_dilate_t));
    }
    for (tile_no = 0; tile_no < R_DK2_TILE_NUM; tile_no++){
        while (drp_lib_status[tile_no] == DRP_NOT_FINISH);
    }
}
```

The number of iterations, current set as 1

Set the input and output addresses according to the number of iterations. The same address was used to overwrite the original image in this sample



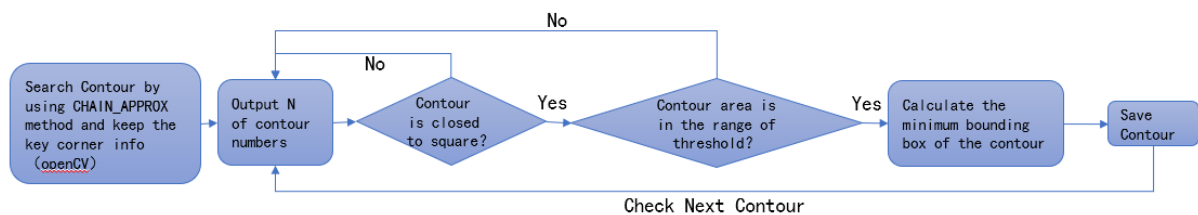
4.5. Pre-Processing step “Find Contour” (OpenCV)

At this point, by calling openCV’s FindContour method to process dilated image, we can get the outer contours of all the highlighted parts in this frame of image and the smallest bounding rectangle of each contour.

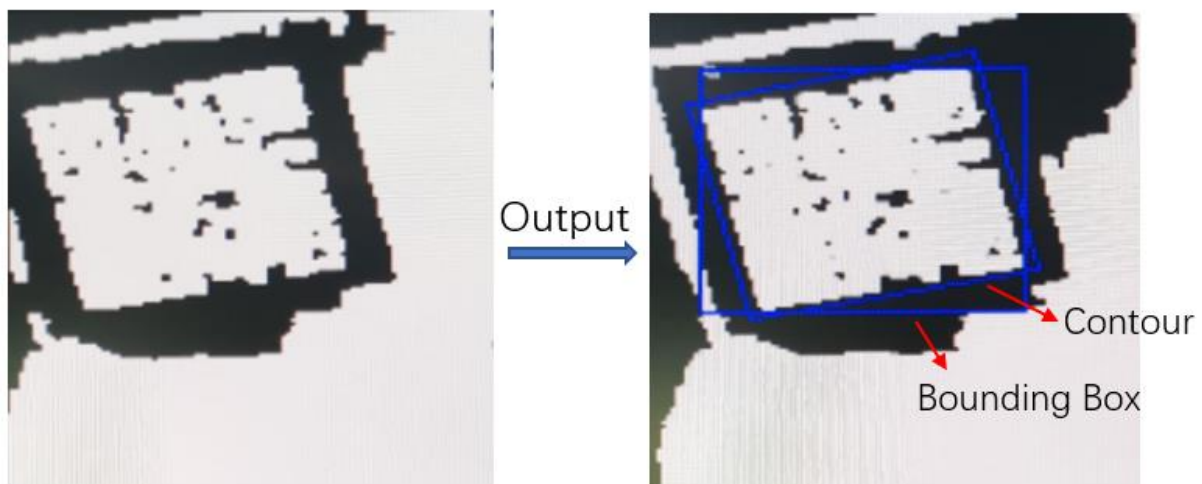
We will determine the aspect ratio, area, location and other information of the contour one by one and filter out the contour most likely to contain the QR code.

If they don’t match, we will sample the next frame.

This step is implemented by the CPU and will take about 4ms.

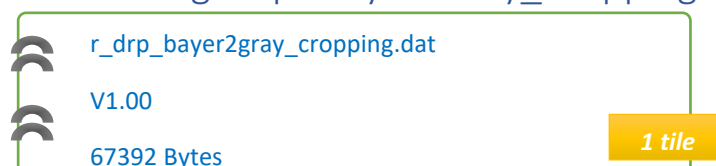


The rectangle with a rotation angle in the figure is the outline of the QR code area. Later we will adjust this rotated rectangle to a graph with an inclination of 0 degrees for finer feature detection.



The coordinates of the four vertices of the smallest bounding rectangle will be converted into the coordinates of the original input image, which are used to crop the area containing the QR code from the original image. Pre-Processing step “Cropping”

4.6. Pre-Processing step “Bayer2Gray_Cropping”



To ensure the accuracy of the identification feature extraction of the QR code, we will crop the original QR code image from the MIPI input buffer.

At this time, we load a bayer2gray_cropping DRP library to crop the image from the MIPI buffer. The cropping position is the coordinates of the 4 vertices of the smallest bounding rectangle in section 4.5.

This bayer2gray_cropping only occupies the hardware resources of 1 Tile, and has the ability to run 6 tiles in parallel, so we will use 6 Tiles DRP for parallel processing.

It will take about 0.6ms for this step, the loading time of DRP is 0.25ms, and the execution time is about 0.35ms



Sample Code

```
ret_val = R_DK2_Load(&g_drp_lib_bayer2gray_cropping[0],
R_DK2_TILE_0 | R_DK2_TILE_1 |
R_DK2_TILE_2 | R_DK2_TILE_3 |
R_DK2_TILE_4 | R_DK2_TILE_5,
R_DK2_TILE_PATTERN_1_1_1_1_1_1, NULL, &cb_drp_finish, &drp_lib_id[0]);
ret_val = R_DK2_Activate(drp_lib_id[TILE_0] | drp_lib_id[TILE_1] |
drp_lib_id[TILE_2] | drp_lib_id[TILE_3] | drp_lib_id[TILE_4] |
drp_lib_id[TILE_5], 0);
```

Load DRP Lib to Tile0~5

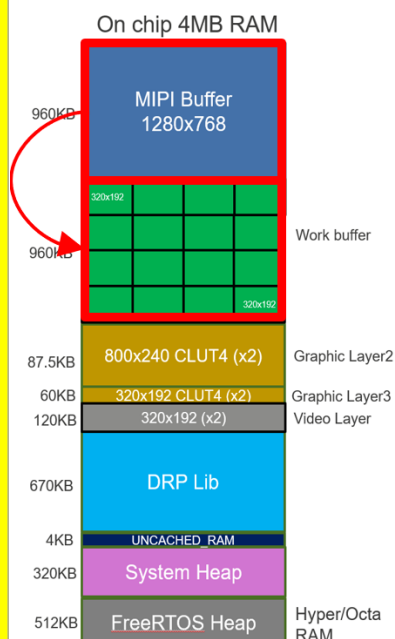
6 Tiles execute in parallel

```
for (tile_no = 0; tile_no < R_DK2_TILE_NUM; tile_no++)
{
/* Set the address of buffer to be read/write by DRP */
param_bayer2gray_cropping.src = input_address;
param_bayer2gray_cropping.accumulate = 0;
param_bayer2gray_cropping.width = input_width;
param_bayer2gray_cropping.height = input_height;
param_bayer2gray_cropping.area_offset_x = offset_x;
param_bayer2gray_cropping.area_width = output_width;
param_bayer2gray_cropping.area_offset_y = offset_y +
tile_no * output_height / R_DK2_TILE_NUM;
param_bayer2gray_cropping.area_height =
output_height / R_DK2_TILE_NUM;
param_bayer2gray_cropping.dst = output_address +
tile_no * output_height /
R_DK2_TILE_NUM * output_width;

drp_lib_status[tile_no] = DRP_NOT_FINISH;

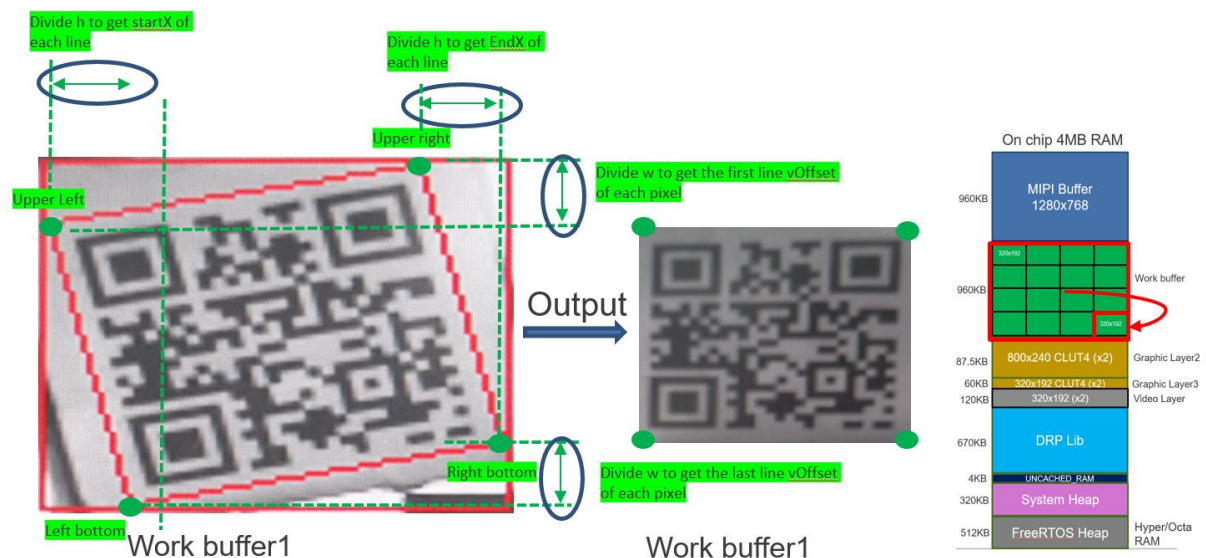
ret_val = R_DK2_Start(drp_lib_id[tile_no],
(void *)&param_bayer2gray_cropping,
sizeof(r_drp_bayer2gray_cropping_t));
DRP_DRV_ASSERT(ret_val);
}

for (tile_no = 0; tile_no < R_DK2_TILE_NUM; tile_no++)
{
while (drp_lib_status[tile_no] == DRP_NOT_FINISH);
}
```



4.7. Pre-Processing step “Keystone”

The left side of the figure below is a cropped grayscale image with a rotation angle, in which 4 green vertices can be obtained from the contour information in section 4.5



In this step, we use CPU to implement a simple Keystone and zoom function to map the images in the 4 green vertices to the image on the right. The keystone correction here is simply to extract or repeat the original pixels without generating new pixels.

Therefore, it is a lossy but fast algorithm.

If you need a more accurate algorithm, you can use the affine function of DRP library, but it will consume more time.

Notes:

- The cropped image is saved in the workbuffer, and the occupied buffer size is dynamic
 - If block#16 of the work buffer is not used in the cropped image:
 - ➔ The result of ladder correction is saved in block16
 - If block#16 of the work buffer is already used to save cropped image
 - ➔ The processing will be terminated and start to process the next frame
- Keystone function performs scaling at the same time as rotation
- The output image size is mapped to several fixed values based on the input size

Input Height	Output Height
<=128	96x96
>128	128x128

4.8. Pre-Processing step “Binarization_Adaptive”

After getting an image without rotation, we repeat the binarization adaptive processing in section 4.3 to obtain a binarization image without rotation angle.

It takes about 0.5ms for this step.

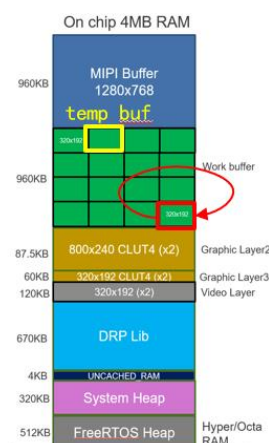


Output

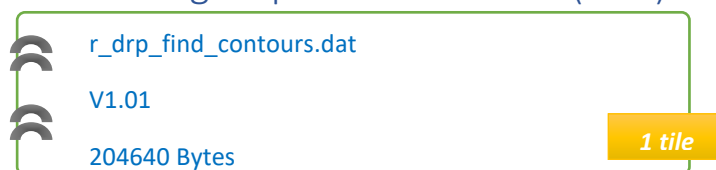


Work Buffer #16

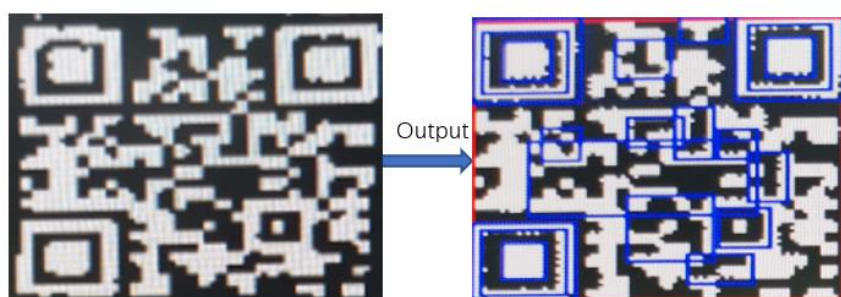
Work buffer #16



4.9. Pre-Processing step “Find Contour” (DRP)



Load the FindContour DRP library at this time to detect the contours of all the highlighted parts in the two-dimensional code image after binarization, and the detection result is the blue rectangular area in the image.



FindContour DRP library will output the location and size data of each contour. All contour information will be used for the next step 2D barcode symbols detection. This process takes about 1.4ms in total, including 0.4ms for DRP library loading and 1.0ms for processing.

The output result looks like in below tables:

Bytes	Contour Rectangle Data						Contour Points Data						
	2	2	2	2	4	4	2	2	2	2	2	2	
Contour1	x	y	W	h	Contour points num	Point to contour points mem	x	y	x	y	Contour1 data set
Contour2	x	y	w	H	Contour points num	Point to contour points mem	x	y	x	y	Contour2 data set
ContourN	ContourN data set

Sample Code

```
ret_val = R_DK2_Load(&g_drp_lib_find_contours[0], R_DK2_TILE_0,
    R_DK2_TILE_PATTERN_2_1_1_1_1,
    NULL, &cb_drp_finish, &drp_lib_id[0]);
DRP_DRV_ASSERT(ret_val);
ret_val = R_DK2_Activate(drp_lib_id[TILE_0], 0);
```

Load DRP Lib to the position of Tile0.
It will use the resource of Tile0 and Tile1

```
R_MMU_VAtoPA((uint32_t*)&param_findc_region[tile_no
    *DRP_R_FINDC_BUFF_REGION_NUM/loop],
    (uint32_t*)&findc_region_bufadr);
```

Where to save the countour points

```
R_MMU_VAtoPA((uint32_t*)&param_findc_rect[tile_no
    *DRP_R_FINDC_BUFF_RECT_NUM/loop],
    (uint32_t*)&findc_rect_bufadr);
```

Where to save the countour rectangles

```
param_findc.src = input_address +
    tile_no*height*input_width;
param_findc.dst_region = findc_region_bufadr;
param_findc.dst_rect = findc_rect_bufadr;
param_findc.work = work_address +
    tile_no*height*input_width;
```

Set temp buffer address,
tile_no is zero in this sample

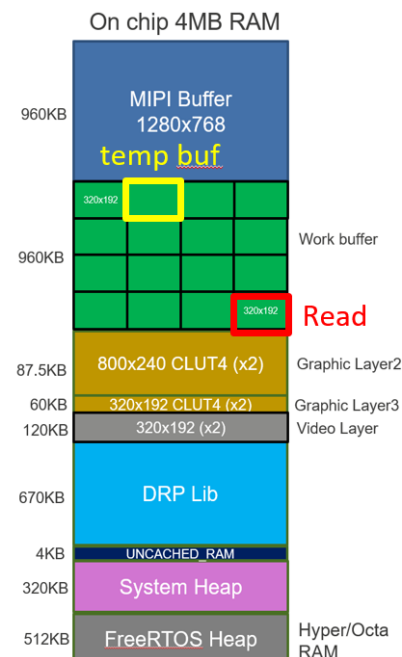
```
param_findc.width = input_width;
param_findc.height = (tile_no == 2) ? (input_height - 2 *
    height) : height;
param_findc.dst_region_size = DRP_R_FINDC_BUFF_REGION_NUM / loop;
param_findc.dst_rect_size = DRP_R_FINDC_BUFF_RECT_NUM / loop;
param_findc.threshold_width = DRP_R_FINDC_WIDTH_THRESHOLD;
param_findc.threshold_height = DRP_R_FINDC_HEIGHT_THRESHOLD;
```

Set maximal counter numbers and minimal counter size

```
drp_lib_status[tile_no*2] = DRP_NOT_FINISH;
ret_val = R_DK2_Start(drp_lib_id[tile_no*2], (void *)&param_findc,
    sizeof(r_drp_find_contours_t));
```

```
while (drp_lib_status[2*tile_no] == DRP_NOT_FINISH){
    if (callback != NULL)
    {
        callback((uint8_t*)input_address, input_width, input_height);
        callback = NULL; //only run once
    }
}
```

DRP find contours process need take about 1ms, we
add a callback function here, when DRP is working,
we can assign a task to CPU, so DRP and CPU can
work in parallel, in this sample, the callback function
will implement DataMatrix symbo detection



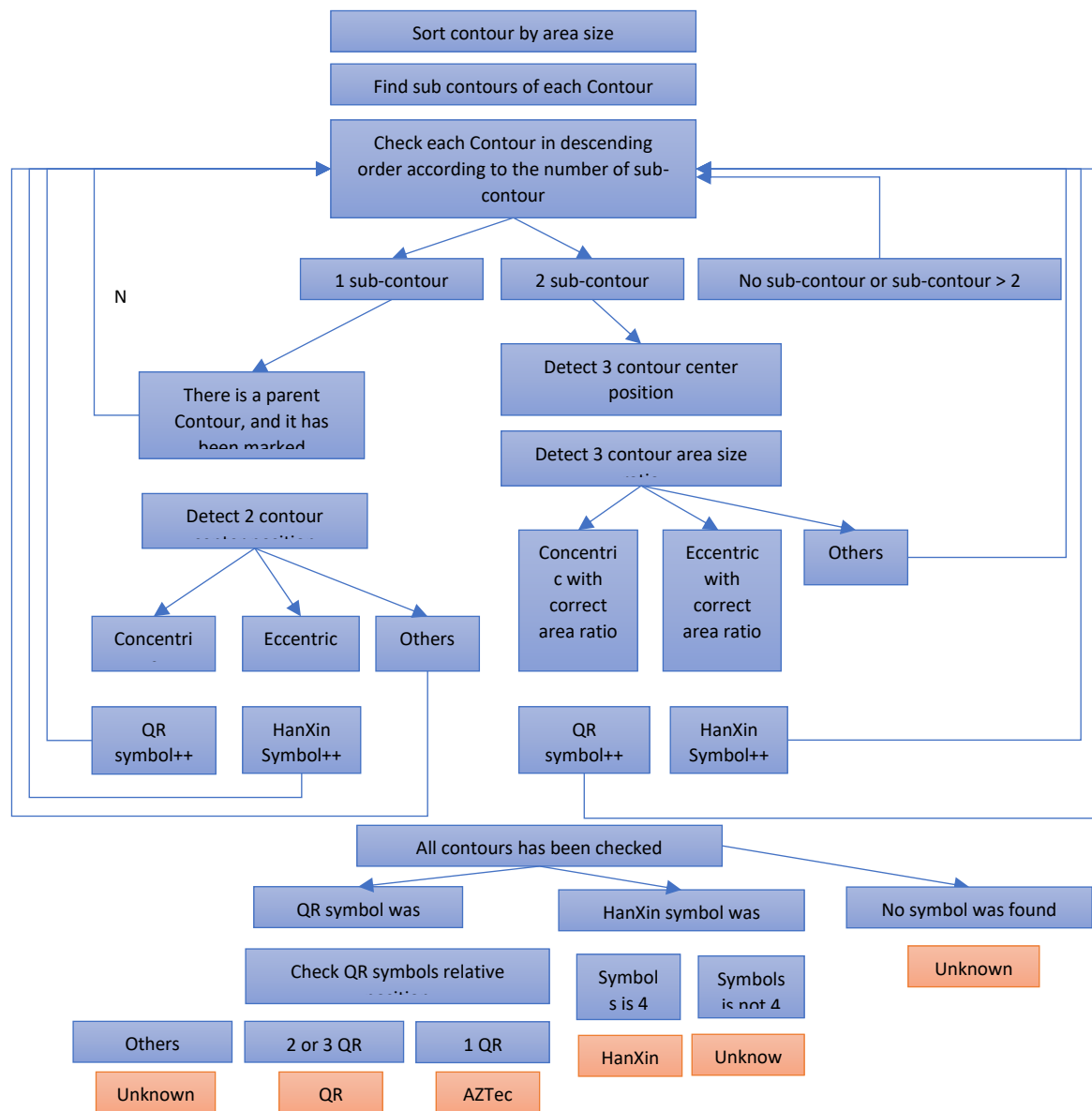
4.10. Pre-Processing step “Barcode Symbol Detection”

Now we have obtained the coordinate information of all contours in the QR code image. We can easily recognize if they are QR code, Micro QR code, Han Xin code, Aztec code by corresponding position of the contour relative to the QR code, whether it is a square contour, contains a symmetrical sub-contour, or contains an eccentric sub-contour.

For DataMatrix detection, we can take advantage of the feature that DRP does not consume CPU load when DRP is running. During the DRP processing in step 8 (1ms DRP execution period), the CPU will calculate the sum statistics of each row and column of the image at the same time, so that enable DRP and CPU run in parallel



The basic flow chart is as follow:



5. Barcode Decoding

In section 4.6, we already got a cropped grayscale barcode image, it only included barcode image, the size is smaller than MIPI input image.

In section 4.10, we already got a barcode type.

Now we can input cropped image and barcode type to ZXing module.

ZXing is able to decode the bar codes with greater efficiency than entering the full MIPI captured image and the unknown bar code type.

```
extern "C" int zxing_decode_image(uint8_t* buf, int width, int height, char * reslut_buf, int reslut_buf_size, barcode_type_t
type)
{
    int decode_result;
    int buf_size = width * height;
    vector<Ref<Result>> results;
    std::string decode_str;
    int size;
    DecodeHints hints;

    switch (type)
    {
        case BARCODE_QR:
            hints = zxing::DecodeHints(DecodeHints::QR_CODE_HINT);
            break;
        case BARCODE_AZTEC:
            hints = zxing::DecodeHints(DecodeHints::AZTEC_HINT);
            break;
        case BARCODE_DATA_MATRIX:
            hints = zxing::DecodeHints(DecodeHints::DATA_MATRIX_HINT);
            break;
        default:
            hints = zxing::DecodeHints(DECODE_HINTS);
    }

    // DecodeHints hints(DECODE_HINTS);
    hints.setTryHarder(false);

    decode_result = ex_decode(buf, buf_size, width, height, true, &results, hints);
}
```

Cropped image in section 4.6

Barcode type in section 4.10

Only enable QR decoder

Only enable AZTEC decoder

Only enable DataMatrix decoder

Unknown type, enable multiple decoders

We didn't provide HanXin and MicroQR decoding in the sample code, you should find the solution from other 3rd parties.





6. Parameter Tuning

Some parameters could be fine tune in r_bcd_main.h

Macro	Default value	Description
DRP_R_BINARY_RANGE	0x1C	The threshold to convert grayscale image to binarization image
DRP_R_FINDC_BUFF_RECT_NUM	60	Maximum number of detectable contours
DRP_R_FINDC_BUFF_REGION_NUM	3	Since we are not using contour points, the output number is set to a small value to reduce memory usage
DRP_R_FINDC_WIDTH_THRESHOLD	5	Only look for contours larger than this threshold width
DRP_R_FINDC_HEIGHT_THRESHOLD	5	Only look for contours larger than this threshold height
KEYSTONE_FIXED_SIZE_ENABLE	disabled	No fixed-size Keystone output is currently used
DRAW_CONTOURS_THRESHOLD_AREA	81	When sorting the contour, the contour with area less than 81 (9x9) is ignored

7. Pre-compiled binary Downloading

In case of using EVK, we provide a flash script to write bootloader and application binaries to QSPI flash. Please connect the JLINK debugger, then double click “program_rza2m_barcode_type.bat” to download the images.

	JLink	2021/8/2 9:32
	Renesas	2021/8/3 11:34
	program_qspi_rza2m.bat	2021/1/6 17:16
	program_rza2m_barcode_type.bat	2021/8/3 11:35

In case of using GR-MANGO, please connect microUSB cable between GR-MANGO and your PC, then copy the rza2m_barcode_type_freertos_gcc_grmango.bin from bin folder to MBED disk.

Some barcode images were used for sample code testing:

