AN13753

Develop the OpenCV Example with MCUXPresso IDE

Rev. 0 — 24 October 2022 Application note

Document information

Information	Content
Keywords	OpenCV, MCU, MCUXPresso IDE
Abstract	OpenCV (Open Source Computer Vision Library) is an open-source library that includes several hundreds of computer vision algorithms.

1 Introduction

OpenCV (Open Source Computer Vision Library: http://opencv.org) is an open-source library that includes hundreds of computer vision algorithms.

OpenCV is released under a BSD license. It is free for both academic and commercial use, designed for computational efficiency. With a strong focus on real-time applications, OpenCV is written in optimized C++ and takes advantage of multicore processors. OpenCV can run under Linux, Windows, and Mac OS X, interfaces for Python, Java, MATLAB, and other languages. It provides a simple-to-use computer vision infrastructure that enables building fairly sophisticated vision applications quickly.

To manipulate images, the OpenCV is not great. It is great for teaching the computer how to see something.

Considering that the OpenCV is usually a PC dedicated computer vision library, which is rare on MCU, we publish this document. This document introduces how to build OpenCV examples on MCUXPresso IDE with OpenCV library. For details about how to build the OpenCV Library, see *Run openCV on Cortex-M7 MCU* (document AN13725). Run it on our RT-Series MCU platform, such as, i.MX RT1170 EVKB board.

2 OpenCV

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

- **Core functionality** - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- **Image processing** - an image-processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- **video** - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- **calib3d** - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- **features2d** - salient feature detectors, descriptors, and descriptor matchers.
- **objdetect** - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars).
- **highgui** - an easy-to-use interface to simple UI capabilities.
- Video I/O - an easy-to-use interface to video capturing and video codecs.
- **gpu** - GPU-accelerated algorithms from different OpenCV modules.
- … some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

All the OpenCV classes and functions are placed into the cv namespace. Therefore, to access this functionality from your code, use the cv: specifier or using namespace cv directive:

```
cv::Mat H = cv::findHomography(points1, points2, CV_RANSAC, 5);
```

Or:

```
using namespace cv;
Mat H = findHomography(points1, points2, CV_RANSAC, 5 );
```

Some of the current or future OpenCV external names may conflict with STL or other libraries. In this case, use explicit namespace specifiers to avoid the conflicts:

```
Mat a(100, 100, CV_32F);
randu(a, Scalar::all(1), Scalar::all(std::rand()));
cv::log(a, a);
a /= std::log(2.);
```

Finally, let us talk about the Mat in OpenCV. If you want to use the OpenCV, the Mat is your first step. The class represents an n-dimensional dense numerical array that can act as a matrix, image, optical flow map, 3-local tensor, and so on.

The public attributes:

Table 1.

attribute	description
MatAllocator* allocator	Custom allocator
int cols	The width of the image
int rows	The height of the image. The cols and rows are (-1, -1) when the matrix has more than two dimensions
uchar* data	Pointer to the data
uchar* dataend	-
uchar* datalimit	-
uchar* datastart	-
int dims	The matrix dimensionality, >=2
int flags	-
int* refcount	Pointer to the reference counter
MSize size	-
MStep step	-

There are many different ways to create **cv::Mat** object. Here are some popular ones:

1. Using cv::Mat::Create(nrows, ncols, type) method or the similar constructor cv::Mat::Mat(nrows, ncols, type[, fill,_vale]) constructor. The **type** has the same meaning, for example, CV_8UC1 means 8-bit single channel matrix and CV_32F2 means 2-channel (that is, complex) floating-point matrix.

   ```
   // make 7x7 complex matrix filled with 1+3j.
   cv::Mat M(7,7,CV_32FC2,Scalar(1,3));
   // and now turn M to 100x60 15-channel 8-bit matrix.
   // The old content will be deallocated
   M.create(100,60,CV_8UC(15));
   ```

2. Use a copy constructor or assignment operator. Matrix assignment is O(1) operation because it only copies the header and increases the reference counter. You can use

the **cv::Mat::clone()** method to get a full (a.k.a. deep) copy of the matrix when you need it.

3. To make a header for user-allocated-data:

```
void init_mat_with_ptr(const unsigned char* pixels,
                         int width, int height, int step)
{
  cv::Mat img(height, width, CV_8UC3, pixels, step);
  cv::GaussianBlur(img, img, cv::Size(7,7), 1.5, 1.5);
}
```

4. Use MATLAB-style matrix initializers, **cv::Mat::zeros()**, **cv::Mat::ones()**, and **cv::Mat::eye()**.

To release the data pointed by a matrix header before the matrix destructor is called, use **cv::Mat::release()**.

The next important thing is how to access the data. The elements are stored in row-major order (row by row). The **cv::Mat::data** member points to the first element of the first row. **cv::Mat::rows** contains the number of matrix rows and **cv::Mat::cols** contains the number of matrix columns. There is yet another member, **cv::Mat::step**, used to actually compute address of a matrix element.

Given these parameters, computer the address of the matrix element, **M_{ij}**, as below:

```
addr(M_{ij})=M.data + M.step*i + j*M.elemSize()
```

If you know the matrix element type, for example, it is float, then you can use **cv::Mat::at()** method:

```
addr(M_{ij})=&M.at<float>(i,j)
```

The reference code is as below:

```
// compute sum of positive matrix elements
// (assuming that M is double-precision matrix)
double sum=0;
for(int i = 0; i < M.rows; i++)
{
    const double* Mi = M.ptr<double>(i);
    for(int j = 0; j < M.cols; j++)
        sum += std::max(Mi[j], 0.);
}
```

3 Create a MCUXPresso project

Make sure that the following items or the newest one have been installed on your PC:

- SDK: 2.11.0 for i.MX RT1170
- MCUXPresso IDE: 11.5.0
- Example: SDK_root\boards\evkmimxrt1170\demo_apps\hello_world_demo_cm7
- The libs: libopencv_world, libopenjp2, libjpeg-turbo, libpng, zlib; generated from the source code of OpenCV according to Run openCV on Cortex-M7 MCU (document AN13725).

To create a MCUXPresso project, perform the following steps:

1. Import the hello world example through the Quickstart Panel:

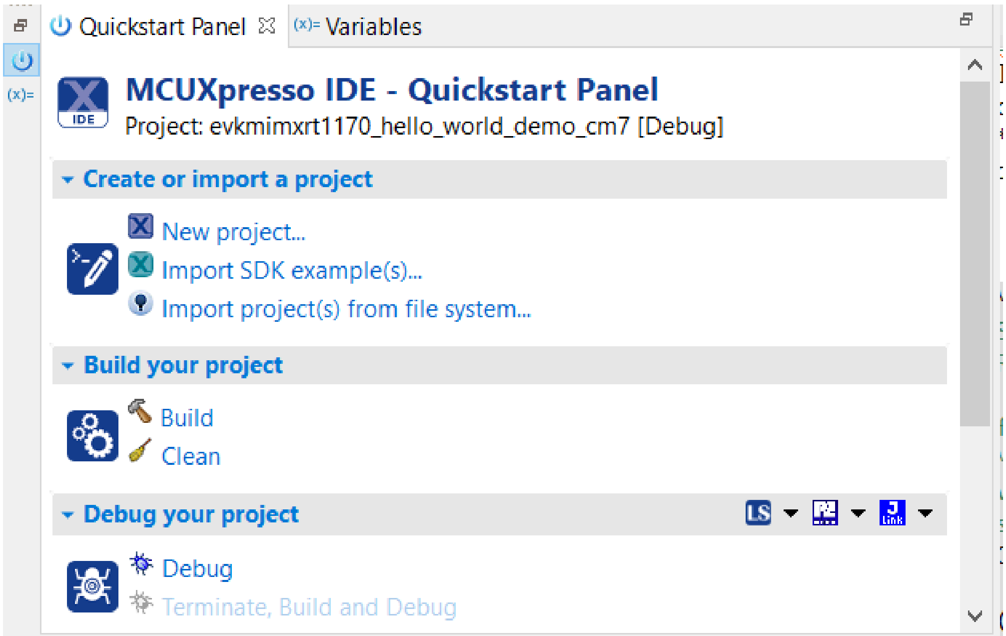

Figure 1. Quickstart panel

2. The OpenCV needs C++, but the *hello_world* example is a **C** project. To edit the project-file to enable the C++ feature, find the *.project* under your workspace and add the below.


```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
	<name>evkmimxrt1170_hello_world_demo_cm7</name>
	<comment></comment>
	<projects>
	</projects>
	<buildSpec>
		<buildCommand>
			<name>org.eclipse.cdt.managedbuilder.core.genmakebuilder</name>
			<triggers>clean,full,incremental,</triggers>
			<arguments>
			</arguments>
		</buildCommand>
		<buildCommand>
			<name>org.eclipse.cdt.managedbuilder.core.ScannerConfigBuilder</name>
			<triggers>full,incremental,</triggers>
			<arguments>
			</arguments>
		</buildCommand>
	</buildSpec>
	<natures>
		<nature>org.eclipse.cdt.core.cnature</nature>
		<nature>org.eclipse.cdt.core.ccnature</nature>
		<nature>com.nxp.mcuxpresso.core.datamodels.sdkNature</nature>
		<nature>org.eclipse.cdt.managedbuilder.core.managedBuildNature</nature>
		<nature>org.eclipse.cdt.managedbuilder.core.ScannerConfigNature</nature>
	</natures>
</projectDescription>
```

Figure 2. C++ supports

Then reopen the project using the MCUXPresso IDE.

3. The project supports the C++ now, but the settings for C++ are empty, only C is required to be configured. The first is for MCU C++ compiler, including the header path and the preprocessor symbols.

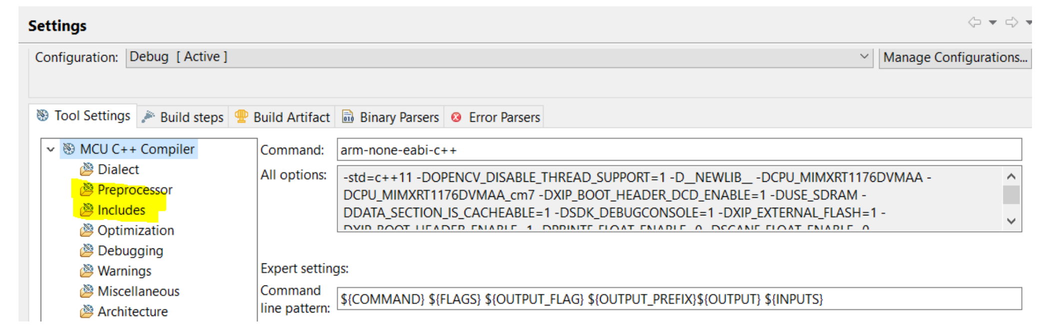

Figure 3. Preprocessor symbol and the includes path

In details, it Includes:

```
"${workspace_loc:/${ProjName}/drivers}"
"${workspace_loc:/${ProjName}/board}"
"${workspace_loc:/${ProjName}/source}"
"${workspace_loc:/${ProjName}/utilities}"
"${workspace_loc:/${ProjName}/drivers}"
"${workspace_loc:/${ProjName}/device}"
"${workspace_loc:/${ProjName}/component/uart}"
"${workspace_loc:/${ProjName}/component/lists}"
"${workspace_loc:/${ProjName}/startup}"
"${workspace_loc:/${ProjName}/xip}"
"${workspace_loc:/${ProjName}/CMSIS}"
"${workspace_loc:/${ProjName}/utilities}"
"${workspace_loc:/${ProjName}/device}"
"your_cv_path\opencv\build"
" your_cv_path \opencv\include"
" your_cv_path \opencv\modules\core\include"
" your_cv_path \opencv\modules\imgcodecs\include"
" your_cv_path \opencv\modules\imgproc\include"
" your_cv_path \opencv\modules\world\include"
" your_cv_path \opencv\modules\highgui\include"
" your_cv_path \opencv\modules\features2d\include"
" your_cv_path \opencv\modules\ml\include"
" your_cv_path \opencv\modules\video\include"
```

The pre-processor symbols include:

```
OPENCV_DISABLE_THREAD_SUPPORT=1
__NEWLIB__
CPU_MIMXRT1176DVMAA
CPU_MIMXRT1176DVMAA_cm7
XIP_BOOT_HEADER_DCD_ENABLE=1
USE_SDRAM
DATA_SECTION_IS_CACHEABLE=1
SDK_DEBUGCONSOLE=1
XIP_EXTERNAL_FLASH=1
XIP_BOOT_HEADER_ENABLE=1
PRINTF_FLOAT_ENABLE=0
SCANF_FLOAT_ENABLE=0
PRINTF_ADVANCED_ENABLE=0
SCANF_ADVANCED_ENABLE=0
FSL_SDK_DRIVER_QUICK_ACCESS_ENABLE=1
MCUXPRESSO_SDK
CR_INTEGER_PRINTF
__MCUXPRESSO
__USE_CMSIS
```

```
DEBUG
```

Configure the MCU C++ Linker, including the Libraries and also the Library search path.

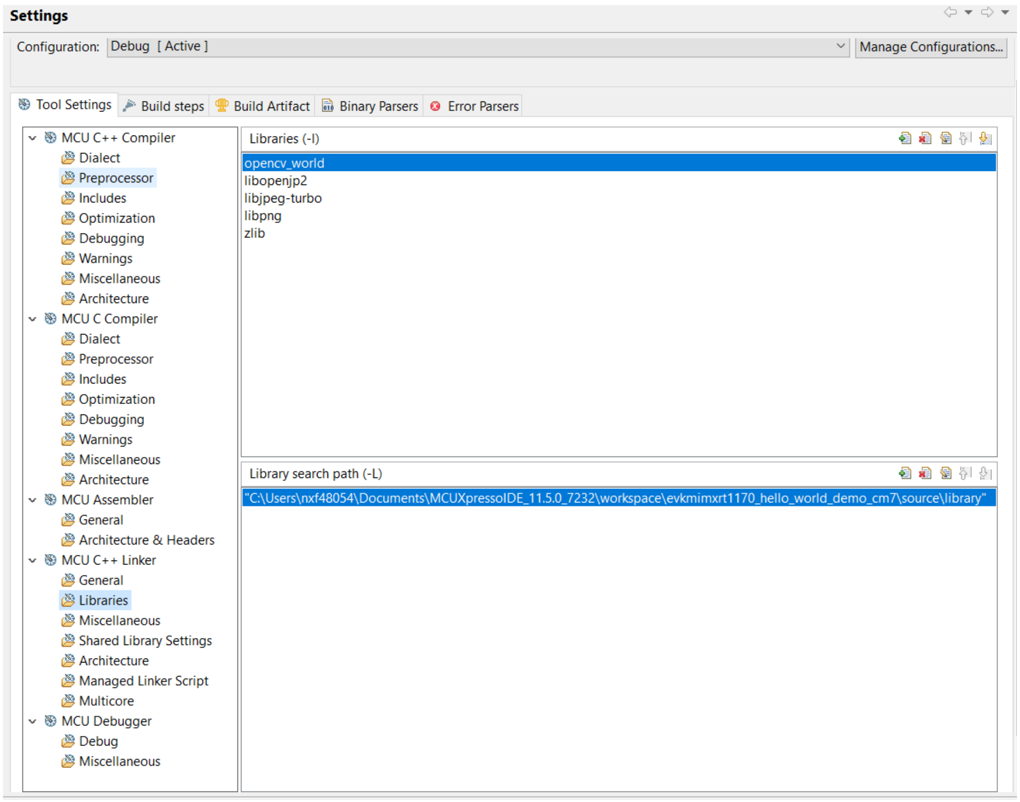

Figure 4. Library configurations

Note that the Library search path is where you place all the libraries.

4. As we know, the OpenCV is written by C++. To call the function, create a C++ file. Simply, we rename the *hello_word.c* to *hello_world.cc* and retain the content.
5. To import the source image, either compressed as *jpeg*, *PNG*, or other raw-data without any compressed. Here we use an ASM instruction, **.incbin**, to achieve this. Create an *asm* file and add it into our project.
   Like this, where you place this file is free, but it is better to place it in the same folder with the *hello_world.cc*:

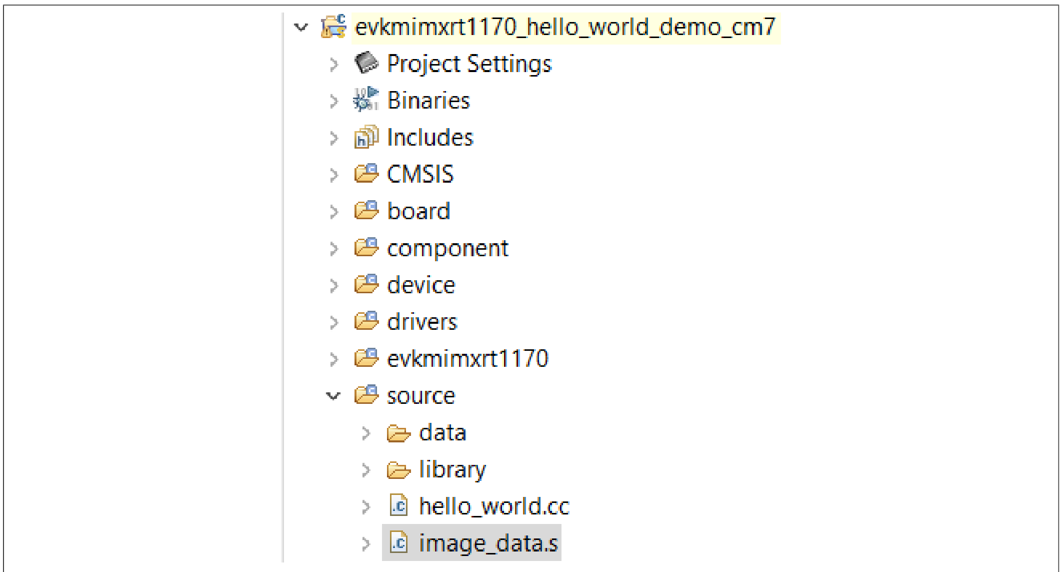

Figure 5. Add the test file

After including this ASM file, edit the file and add:

```
    .global img_start
    .global img_end

img_start:
  .incbin "data/lena.jpg"
img_end:
```

You can change the image to any you like. But if you want to use the Relative path, the IDE finds the file from where you place the *hello_world.cc*, which means that you must place all your *image_data* or *image_folder* to the same folder with *hello_world.cc*. If not, the IDE cannot find the data.

6. Now the project of MCUXPresso is ready. To validate the project, write some code and develop some examples.

4 Deploy some OpenCV examples on MIMXRT1170 EVK

This chapter introduces the code snippet about how to develop some OpenCV examples. All the code can be found in the attachment. To call the code from the *hello_word.cc*, you can either put the code into *hello_world.cc* or align them to a new C++ file. Also if you do not want to rename the *hello_world.c* to *hello_world.cc*, do not forget to use the extern **C** to declare your functions. Otherwise, the link error occurs.

1. To include the header, we only need one line, which is so friendly and handy.

```
#include "opencv2\opencv.hpp"
```

2. The OpenCV use the cv::Mat to organize the data, so first we need to declare and define the input data and create a *cv::Mat* instance. Consider that we do not have a filesystem, so we use an asm-instruction, **.incbin**, to import the picture. As we have defined it in the previous chapter, we can use the symbol here. If the picture is compressed, decode them to the process. So we can read the data from memory and then call OpenCV to do the decoding. If the picture is raw-data, we can use it directly:

```
extern uint8_t img_start[];
```

```
extern uint8_t img_end[];
#define IMG_LEN (img_end - img_start)
// compressed data
std::vector<char> data(img_start, img_start + IMG_LEN);
cv::Mat img = cv::imdecode(cv::Mat(data), IMREAD_UNCHANGED);
// raw data, need to aware the shape, and also the depth,
 such as rgb == CV_8UC3, equal to
// each pixel has 3 items, and each item is 8bits
Mat img(Size(480, 360), CV_8UC3);
memcpy(img.data, img_start, IMG_LEN);
```

3. Find contours and draw the contours:

```
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(dst, contours, hierarchy, RETR_EXTERNAL,
 CHAIN_APPROX_SIMPLE);
// To display the contours
Mat resultImage = Mat ::zeros(dst.size(),CV_8U);
drawContours(resultImage, contours, -1, Scalar(255, 0, 255));
```

4. Now we perform a complex task to find squares. The challenge is to find all the squares of a given picture, as shown in Figure 6.

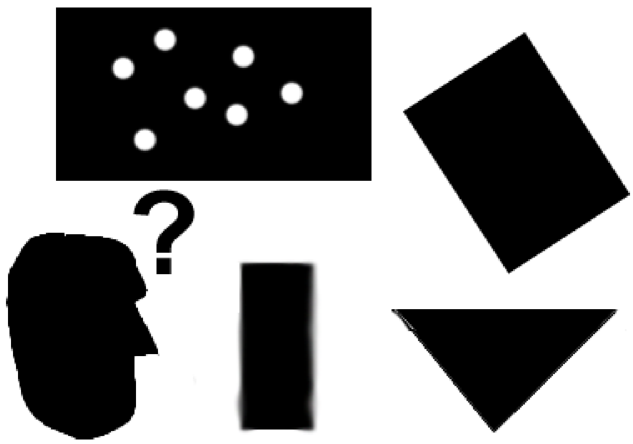

Figure 6. Test data

The code snippet is below with multiple cv APIs, and the result is pushed into a vector:

```
// returns sequence of squares detected on the image.
static void findSquares( const Mat& image,
 vector<vector<Point> >& squares )
{
    squares.clear();
    Mat pyr, timg, gray0(image.size(), CV_8U), gray;
    // down-scale and upscale the image to filter out the
 noise
    pyrDown(image, pyr, Size(image.cols/2, image.rows/2));
    pyrUp(pyr, timg, image.size());
    vector<vector<Point> > contours;
    for( int c = 0; c < 3; c++ )
    {
      int ch[] = {c, 0};
      mixChannels(&timg, 1, &gray0, 1, ch, 1);
```

```
      // try several threshold levels
      for( int l = 0; l < N; l++ )
      {
        if( l == 0 )
        {
          Canny(gray0, gray, 0, thresh, 5);
          dilate(gray, gray, Mat(), Point(-1,-1));
        }
        else
        {
          gray = gray0 >= (l+1)*255/N;
        }
        findContours(gray, contours, RETR_LIST,
 CHAIN_APPROX_SIMPLE);
        vector<Point> approx;

        // test each contour
        for( size_t i = 0; i < contours.size(); i++ )
        {
        // approximate contour with accuracy proportional to
 the contour perimeter
        approxPolyDP(contours[i], approx,
 arcLength(contours[i], true)*0.02, true);
        // square contours should have 4 vertices after
 approximation
        if( approx.size() == 4 &&
          fabs(contourArea(approx)) > 1000 &&
          isContourConvex(approx) )
        {
          double maxCosine = 0;

          for( int j = 2; j < 5; j++ )
          {
            // find the maximum cosine of the angle between
 joint edges
            double cosine = fabs(angle(approx[j%4],
 approx[j-2], approx[j-1]));
            maxCosine = MAX(maxCosine, cosine);
          }
          if( maxCosine < 0.3 )
            squares.push_back(approx);
          }
        }
      }
    }
}
```

5. To encode a raw-data to specified format, if you have a filesystem, get the data and then write it to a file. If not, maybe you can download the memory to your PC and check the result.

```
std::vector<uchar> decoded_img;
cv::imencode(".jpeg", img, decoded_img);
uchar *data = decoded_img.data();
```

Pay attention to the first parameter of the **cv::imencode**. Do not forget the **.** before the format. It is **.jpeg** and not **jpeg**.

5 Reference

The files mentioned in the article are shipped in the attachments.

- https://vovkos.github.io/doxyrest-showcase/opencv/sphinx_rtd_theme/index.html#doxid-d1-dfb-intro
- https://physics.nyu.edu/grierlab/manuals/opencv/classcv_1_1Mat.html

6 Revision history

Rev.	Date	Description
0	20 October 2022	Initial release

7 Legal information

7.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

7.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

7.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

Date of release: 24 October 2022
Document identifier: AN13753