# AN14287

## Machine Inspection Demo for i.MX 93

Rev. 1 — 22 April 2024 Application note

**Document information**

| Information | Content |
|---|---|
| Keywords | TSN, Real-Time-Edge, robot control, computer vision, object detection |
| Abstract | This document describes the Machine inspection demo showcasing TSN and ML capabilities of the i.MX 93 and provides setup instructions. |

# 1 Introduction

The Real-time Edge software enables real-time applications for i.MX products. It supports advanced real-time network protocols and stacks for Time-Sensitive Networking (TSN) and PTP network synchronization.

This demonstration uses the GenAVB software stack to control a robot arm via a TSN network and to ensure that the system fulfills the real-time requirements of the application. The use case described in this application note simulates a conveyor belt where a camera scans the products on the belt, and a robot arm selects the correct product. For this, a tablet simulates the conveyor by displaying passing oranges and apples. The i.MX 93 continuously records the display of the tablet using an onsemi AP1302 camera and processes the captured frames to determine if the passing fruits are oranges or apples. This fruit recognition is done using a TensorFlow Lite neural network running on the Neural Processing Unit of i.MX 93. Once the coordinates of an apple are determined, the commands are sent from the i.MX 93 into the TSN network composed of an i.MX RT 1180 TSN switch and i.MX RT 1170 TSN endpoint. Upon reception of the commands by the i.MX RT 1170, it controls the robot arm through the UART interface to touch the apple. All three boards in the TSN network are configured to use Qbv settings for Quality of Service (QOS).

# 2 Hardware setup

This section gives the details of the hardware setup.

## 2.1 Necessary hardware

The setup is composed of 4 main elements:

- The i.MX 93 EVK with camera and accessories:
  - i.MX93 EVK with A1 SoC revision
  - ON Semiconductor AP1302 camera module
  - IMX-MIPI-HDMI adapter to connect an HDMI display
  - USB Keyboard
- The i.MX RT1180 EVK as a TSN switch
- The i.MX RT1170 EVK and the robot arm:
  - MIMXRT1170-EVK board
  - uARM SwiftPro robot arm equipped with its capacitive stylus
- A touchscreen-capable display to simulate the conveyor belt validated with a Dell touchscreen monitor and a Galaxy Tab A (SM-T510)

## 2.2 Connections and reworks

Figure 1 illustrates the connections between each component.

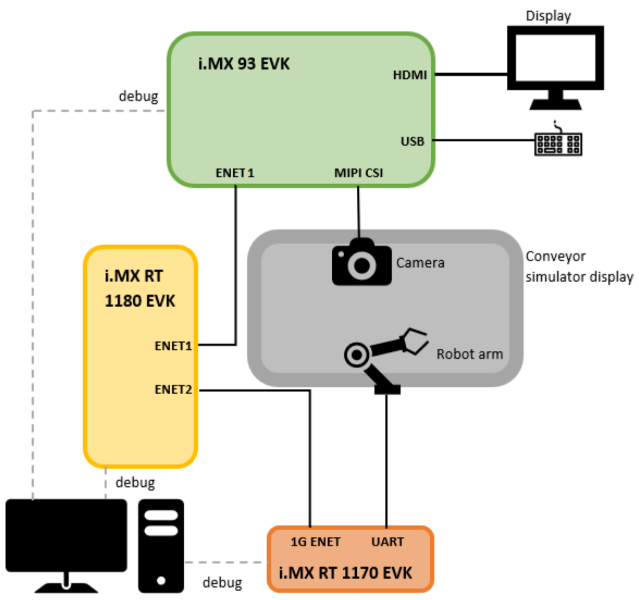

**Figure 1. Hardware setup**

### 2.2.1 i.MX 93 EVK

The i.MX 93 EVK is connected to the camera via the CSI interface, to its display via the IMX-MIPI-HDMI adapter, and to the i.MX RT 1180 via the TSN-capable connector ENET1 (mapped to eth1).

### 2.2.2 i.MX RT 1180 EVK

The i.MX RT 1180 switch is connected to the i.MX 93 EVK via the ENET1 port and to the i.MX RT 1170 EVK via the ENET2 port.

### 2.2.3 i.MX RT 1170

The i.MX RT 1170 EVK is connected to the TSN network using the 1G ENET port and to the robot arm through the UART interface (pins 2 and 4 of the J9 connector plus ground). Figure 2 and Figure 3 show the connection between the i.MX RT 1170 EVK and the robot arm.

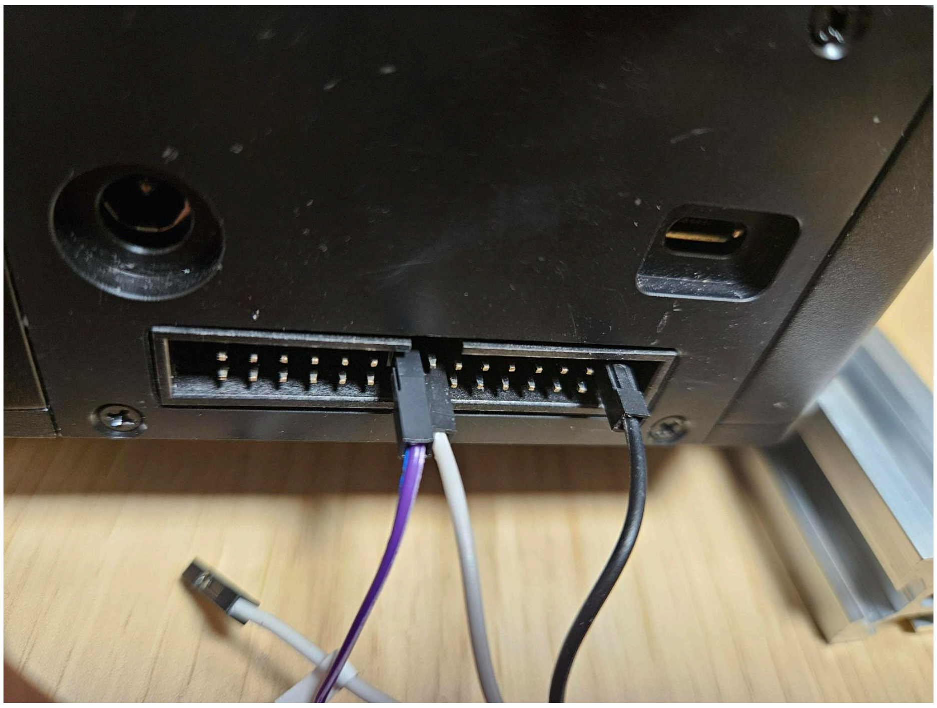
Figure 2. Robot arm UART connection

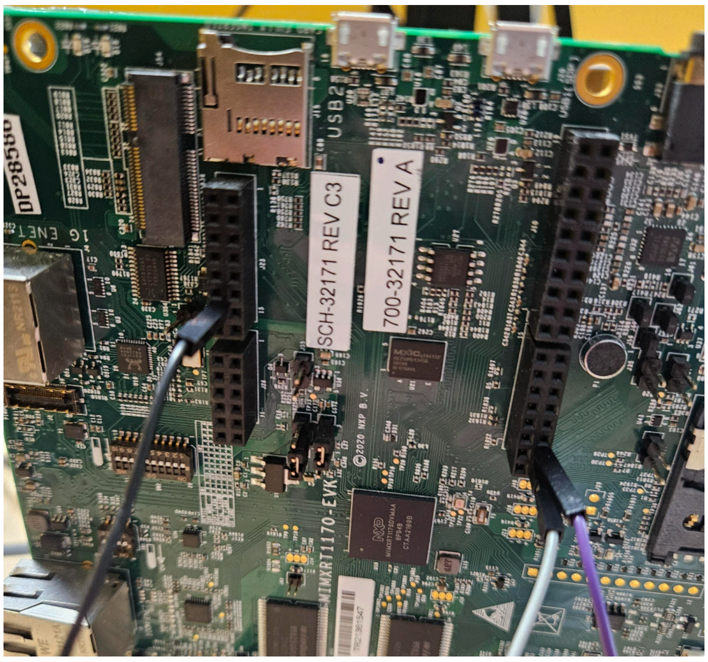

Figure 3. i.MX RT 1170 UART connection

***Note:*** *The i.MX RT 1170 UART works at 3.3 V, while the robot arm at 5 V. We highly recommend to install the following setup between them:*

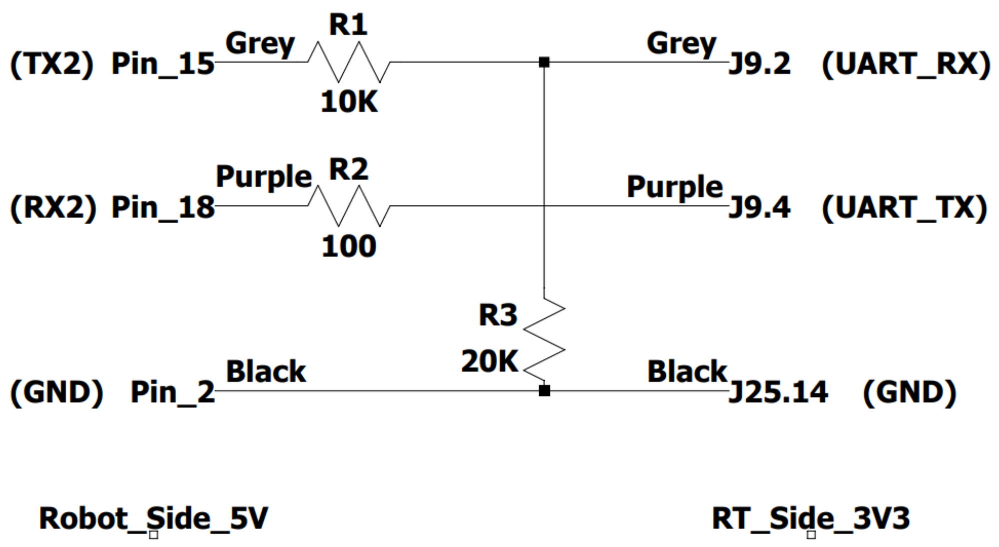

*Figure 4. Resistors to add between i.MX RT 1170 and Robot arm*

When the BLE chip is mounted on the i.MX RT 1170 EVK (U16), it uses UART2 to communicate with the BLE chip, so this connection must be removed to connect this UART to the robot arm. To do it, resistors R798 and R799 must be removed. Figure 5 shows the placement of these resistors.

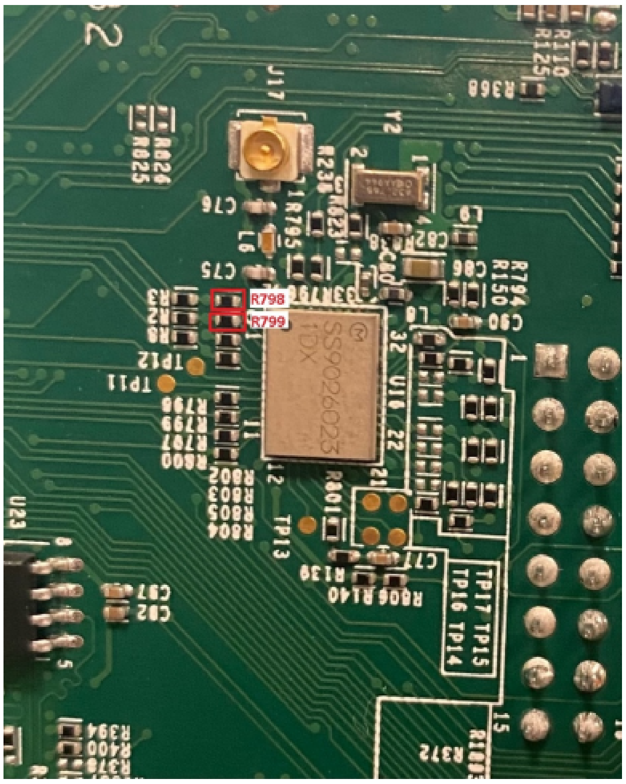

**Figure 5. i.MX RT 1170 resistors to remove**

## 3 Software setup

This section gives the details of the software setup.

First, clone the repo containing the meta layer for Yocto and the patches for the i.MX RT 1170:

```
$ git clone https://github.com/nxp-imx-support/machine-inspection-demo.git
```

Then, check out the `v2_7-2_11_0-4_3_0_imx93` tag.

### 3.1 i.MX 93 setup

The image running on the i.MX 93 EVK is based on the Real-Time Edge 2.7 Yocto release. An additional meta layer “meta-machine-inspection-demo” is used to add a new distro that:

- Includes eIQ packages to use the NPU for the image detection algorithm
- Adds required Python and graphical packages
- Adds support for X11
- Adds the GenAVB serial controller profile
- Adds the demonstration files (Python script, neural network model, and uARM SDK)

The following section describes the build process starting with the creation of a “machine-inspection-demo” folder to contain all the build elements:

```
$ mkdir machine-inspection-demo
$ cd machine-inspection-demo/
```

1. Install the repo utility:

   ```
   $ mkdir ~/bin
   $ curl https://storage.googleapis.com/git-repo-downloads/repo  > ~/bin/repo
   $ chmod a+x ~/bin/repo
   $ export PATH=${PATH}:~/bin
   ```

2. Initialize the Yocto build based on the Real-Time Edge v2.7 Yocto release:

   ```
   $ mkdir yocto-real-time-edge
   $ cd yocto-real-time-edge
   $ repo init -u https://github.com/nxp-real-time-edge-sw/yocto-real-time-
   edge.git -b real-time-edge-mickledore -m real-time-edge-2.7.0.xml
   $ repo sync
   ```

3. Copy the meta-machine-inspection-demo layer from the cloned repository in section 3 to the sources folder in the build folder:

   ```
   $ cp -r <path_to_cloned_repository>/meta-machine-inspection-demo
   <path_to_machine_inspection_demo_folder>yocto-real-time-edge/sources/
   ```

4. Set up a build folder for the Machine Inspection distro:

   ```
   $  DISTRO=nxp-real-time-edge-machine-inspection MACHINE=imx93evk source real-
   time-edge-setup-env.sh -b build-machine-inspection
   ```

5. Add the machine inspection layer to `conf/bblayers.conf` by appending it to the list of bblayers:

   ```
   [In build-machine-inspection/conf/bblayers.conf ]
   BBLAYERS += "${BSPDIR}/sources/meta-machine-inspection-demo"
   ```

6. Build the image:

   ```
   $  bitbake nxp-image-real-time-edge
   ```

   After the build, the final image will be placed at `<build_directory>/tmp/deploy/images` and can be flashed as follows:

   ```
   $ unzstd -f <image_name>wic.zst
   $ sudo dd if=<image_name>.wic of=/dev/sd<partition> bs=1M conv=fsync
   ```

   Where <partition> is the letter of the SD card device in Linux.

***Note:*** *Prerequisites and the detailed explanation of the Real-Time Edge Yocto build can be found in the Real-time Edge Yocto Project User Guide (document RTEDGEYOCTOUG).*

## 3.2 i.MX RT 1180 setup

The image running on the i.MX RT 1180 EVK is based on the GenAVB/TSN MCUXpresso SDK version 5.11 and the MCUXpresso SDK version 2.14.2.

To get access to the i.MX RT 1180 GenAVB/TSN MCUXpresso release, contact your NXP representative.

The MCUXpresso SDK can be downloaded from https://mcuxpresso.nxp.com

The NXP GenAVB/TSN MCUXpresso User's Guide under `genavb_tsn-mcuxpresso-SDK_2_14_2-5_11_0/doc/` describes the needed MCUXpresso configuration, how to set up the build environment and how to build the i.MX RT 1180 TSN example application. For the bridge application, the cm33 variant of the tsn-app must be built.

For instructions on flashing the binary to the board, refer to chapter 2.2 of the GenAVB/TSN Stack FreeRTOS Evaluation User Guide under `genavb_tsn-mcuxpresso-SDK_2_14_2-5_11_0/doc/`.

***Note:*** *The build script used for this demonstration is* `build_release.sh`.

## 3.3 i.MX RT 1170 setup

The image running on the i.MX RT 1170 EVK is based on the genAVB/TSN MCUXpresso SDK version 4.3.0 and the MCUXpresso SDK version 2.11.0.

The GenAVB release under the name “AVB/TSN stacks for supported i.MX RT crossover MCUs” can be downloaded from the NXP website and the MCUXpresso SDK from https://mcuxpresso.nxp.com.

The NXP GenAVB/TSN MCUXpresso User's Guide under `genavb_tsn-mcuxpresso-SDK_2_11_0-4_3_0/doc/` describes the needed MCUXpresso configuration, how to set up the build environment, and how to build the i.MX RT 1170 TSN example application. This document also includes the required middleware and version for the MCUXpresso SDK build in section 6.3. To setup your build environment, follow sections 4-8.

***Note:*** *The build script used for this demonstration is* `build_release_serial.sh`.

In addition to the patches for GenAVB, add the following patches to the MCUXpresso SDK for the machine inspection demo:

```
$ cd <path-to-MCUXpresso-SDK>
$ cp <path-to-machine-inspection-demo>/sdk_patches/000*.
$ patch -p0 < 0001-Don-t-use-component-IRQ-handler.patch
$ patch -p0 < 0002-Adapt-driver-IRQ-handler.patch
```

The default configuration for the TSN application must be changed:

```
$ cd <path-to-GenAVB-release>/genavb_tsn-mcuxpresso-SDK_2_11_0-4_3_0/genavb-
appsfreertos-4_3_0/
```

```
$ cp <path-to-machine-inspection-demo>/apps_patches/000* .
$ patch -p0 < 0001-Change-default-config-to-serial-mode.patch
```

An additional patch is needed if you are using i.MX RT 1170 EVK Rev B:

```
$ cd <path-to-GenAVB-release>/genavb_tsn-mcuxpresso-SDK_2_11_0-4_3_0/
SDK_2_11_0_MIMXRT1170-EVK/
$ cp <path-to-machine-inspection-demo>/apps_patches/000* .
$ patch -p0 < 0002-apps-Select-Rev-B-board.patch
```

Now, the image can be built:

```
$ cd boards/evkmimxrt1170/demo_apps/avb_tsn/tsn_app/argmcc
$ export ARMGCC_DIR=<path_to_gcc_toolchain>
$ ./build_release_serial.sh
```

***Note:*** *If you get any compile errors regarding the “PRIx64” usage in `genavb.c`, comment or remove that line in genavb.c*

For instructions on flashing the binary to the board, refer to chapter 2.2 of the GenAVB/TSN Stack FreeRTOS Evaluation User Guide under `genavb_tsn-mcuxpresso-SDK_2_11_0-4_2_0/doc/`.

***Note:*** *Prerequisites and detailed explanation on the GenAVB build can be found in the NXP GenAVB/TSN MCUXpresso User Guide (document GAVBMCUXUGM) that is is available upon request.*

## 3.4 Conveyor display setup

This section provides information on the conveyor display setup.

***Note:*** *In both setups, the orientation of the display is important. The fruits must be passing from left to right from the robot’s point of view and bottom to top from the camera’s point of view.*

### 3.4.1 Windows host

These are the steps for setting up the conveyor on a Windows 10 host with a touchscreen-enabled monitor connected to the host and placed underneath the motor arm and the camera:

1. Obtain Apache Server version >= 2.4.55 from the Apache website and extract the archive.
2. Launch an administrator Command Prompt and install the server:
   ```
   > cd <extract_folder>\Apache24\bin\
   > httpd -k install
   ```
3. Open the Task Manager, go to the Services tab and look for *Apache2.4*. If it is not running, right-click it and click **Start.**
4. Check that the browser is now running by accessing `http://localhost` in a web browser.
5. Download the conveyor belt HTML code from https://github.com/NXP/imx-machine-inspection-assets and copy the contents of the `conveyor` directory to `Apache2.4/htdocs`.
6. Open a browser window and access http://localhost/conveyor-3d.html . The conveyor interface appears. Move the browser window to the touchscreen monitor and change the screen orientation in Display Settings as described in the note above.

### 3.4.2 Android tablet

Alternatively, an Android tablet can be used as the conveyor display using the following steps:

1. Download the KWS application from the play store.

2. Copy the conveyor folder from https://github.com/NXP/imx-machine-inspection-assets to the download folder of the tablet.
3. Open the KWS application and navigate to the **Settings** tab.
4. In the ADVANCED section, enable the Directory index.
5. Start the KWS server.
6. Open the tablet browser (Chrome, for example).
7. On the browser URL bar, type the server's IP address followed by the port number and the path to the conveyor folder. For example: http://192.168.0.20:8080/Download/conveyor.
8. Select the `conveyor-3d.html` file.

### 3.5 Robot arm setup

This demonstration was developed using the UFACTORY uArm Swift Pro firmware 4.9.0. It can be downloaded and flashed through the UFACTORY uArm Studio: https://www.ufactory.cc/pages/download-uarm.

# 4 Running the demo

Once the Software and Hardware setups are ready, power on all 3 boards and the robot arm. Then follow the steps below to run the demo.

## 4.1 First-time setup for i.MX 93

The following steps must be done only the first time after flashing the boards, to install the camera firmware and prepare the ML model to run on the NPU:

### 4.1.1 Install AP1302 camera firmware

1. Download ap1302_60fps_ar0144_27M_2Lane_awb_tuning.bin from OnSemi GitHub NXP_i.MX93_ap1302_firmware by following README.md.
2. Rename it as `ap1302.fw`.
3. Copy it to the target board under `/lib/firmware/imx/camera`.

### 4.1.2 Prepare model

Connect to the board via serial. Then, change to the directory to where the model is located and convert it using vela:

```
cd /home/root/machine-inspection/machine-inspection-demo/
vela ssdlite_mobiledet_dsp_320x320_coco_2020_05_19.tflite
cp output/ssdlite_mobiledet_dsp_320x320_coco_2020_05_19_vela.tflite model.tflite
```

## 4.2 i.MX RT 1180 instructions

For more details about the configurations in this section, refer to chapters 5, 5.2 and 6 of the GenAVB/TSN Stack FreeRTOS Evaluation User's Guide under `genavb_tsn-mcuxpresso-SDK_2_14_2-5_11_0/doc/`.

1. Connect to the serial port of the i.MX RT 1180.
2. Create VLAN entries for ports ENET1 and ENET2 mapped respectively to logical ports 3 and 4 as follows:

```
BRIDGE>> vlan_update 2 3 -c 1 -p
BRIDGE>> vlan_update 2 4 -c 1 -p
```

3. Create FDB entries for ports ENET1 and ENET2 mapped respectively to logical ports 3 and 4 as follows:

```
BRIDGE>> fdb_update 91:e0:f0:00:fe:70 2 4 -c 1 -p
BRIDGE>> fdb_update 91:e0:f0:00:fe:71 2 3 -c 1 -p
```

4. Create Scheduled Traffic entries per port as follows for ports 3 and 4.

```
BRIDGE>> cd /
BRIDGE>> qbv_set 3 -b 1500000 -c 2000000 -C 0 -l 20,400000 -l df,1600000 -p
BRIDGE>> qbv_set 4 -b 1500000 -c 2000000 -C 0 -l 20,400000 -l df,1600000 -p
```

## 4.3 i.MX 93 instructions

All necessary configurations are done directly by the machine inspection script, so launch the machine inspection demo:

```
cd machine-inspection/machine-inspection-demo
chmod a+x run.sh
./run.sh
```

The camera video must be streaming on the connected display and the robot arm must move to the standby position and then to the rest position.

## 4.4 Machine inspection demo instructions

This section gives detailed instruction on the machine inspection.

### 4.4.1 Calibration

The application starts in the calibration state. The objective of this state is to calculate the relation between the coordinates measured through the camera and the ones from the robot.

The calibration is done as follows:

1. Select the **Calib** tab in the conveyor display.
2. Press **f** on the board’s keyboard to find the apple, the video stream from the board should show a black cross on the apple.
3. Move the robot arm so that the point touches the center of the cross.
4. Press the space bar to register the coordinates.
5. Move the apple to a different position (ideally, another corner).
6. Follow steps 2 to 4 three more times (4 times in total) to finish the calibration.

If a new calibration is needed, press **c** on the board’s keyboard to change to the calibration state and then press **r** to erase the previous calibration.

### 4.4.2 Validation

The validation state allows verifying the calibration previously done by hovering the robot arm on top of the apple. This state is entered by pressing **v** on the board’s keyboard.

The apples are automatically detected and the robot arm moves on top of them when the space bar is pressed.

### 4.4.3 Play

The machine inspection demonstration is fully running in the play state. Here, the oranges and apples are moving in the conveyor simulator, the i.MX 93 is detecting the apples and the robot arm is clicking them to make them disappear.

To do it, select the **>||** tab in the tablet and press **p** on the board's keyboard. Pressing **p** in play state switches the application between play and pause mode.

## 4.5 Stressing the network

The goal of this chapter is to highlight the TSN configuration of the network which ensures that the robot commands and responses are correctly delivered in time.

### 4.5.1 Network without TSN

The first test consists of running the machine inspection demo without any TSN configuration and verifying if it is able to handle additional best-effort traffic.

On the i.MX 93, the `run.sh` script must be modified to remove the Qbv configuration:

```
#! /bin/bash

export DISPLAY=:0.0

ifconfig eth1 up
ip link add link eth1 type vlan id 2
#tc qdisc replace dev eth1 parent root handle 100 taprio \
# num_tc 3 \
# map 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 \
# queues 1@0 1@1 1@2 \
# base-time 000500000 \
# sched-entry S 0x2 400000 \
# sched-entry S 0x5 1600000 \
# flags 0x2
```

The demonstration can now be started as usual. Since the network is only used by the demonstration application, the robot must be able to touch the apples as intended.

To stress the network, use the *Real-time Edge Software User Guide* (document REALTIMEEDGEUG ) section 6.2.3.6.5.4 "Scheduled traffic evaluation with TX best-effort traffic":

1. Connect a personal computer or another i.MX board to the ENET3 port of the i.MX RT 1180
2. Add the following configuration to the i.MX RT 1180 bridge console:

   ```
   BRIDGE>> vlan_update 2 5 -c 1 -p
   BRIDGE>> fdb_update 91:e0:f0:00:fe:72 2 5 -c 1 -p
   BRIDGE>> qbv_set 5 -b 1500000 -c 2000000 -C 0 -l 20,400000 -l df,1600000 -p
   ```

3. Run `iperf3` in server mode on the PC (replace ethX by the PC interface connected to the RT 1180):

   ```
   ifconfig ethX 192.168.1.10 up
   iperf3 -s &
   iperf3 -s -p 5202 &
   ```

4. Run `iperf3` in client mode on the i.MX 93:

   ```
   ifconfig eth1 192.168.1.80
   taskset 1 iperf3 -c 192.168.1.10 -u -b 900m -i 2 -t 100 &
   ```

```
taskset 2 iperf3 -p 5202 -c 192.168.1.10 -u -b 0 -i 2 -t 100 &
```

In this scenario, the `iperf3` traffic stresses the network and controls that the traffic to and from the robot arm does not traverse the network with low latency. As a result, the robot is not able to touch the apples as before. In the i.MX 93 console, the log message `.swift.set_position` returned error TIMEOUT with wait=True signals that the communication with the robot is timed-out because the packets to and from the robot arm did not arrive on time.

### 4.5.2 Network with TSN

The same test can be performed with the TSN configuration. To do it, the default `run.sh` script on the i.MX 93 must be used. The same iperf3 commands can be used for the test.

In this case, the TSN Qbv schedule reserves bandwidth for the robot control traffic. This reserved bandwidth is not available to the best-effort traffic, so the control traffic is passed across the network with low latency. As a result, the robot arm is able to touch the apples at the correct time.

# 5 Technologies used

This demo highlights the gPTP stack and the TSN example application provided by NXP's GenAVB software. This software in addition to the Quality Of Service (QOS) configuration ensures the synchronization of the three boards and guarantees the real-time requirements. For more detailed information on GenAVB's configuration and log files, refer to the *Real-time Edge Software User Guide* (document REALTIMEEDGEUG).

The BSP built for this demonstration also includes NXP's eIQ software which allows the neural network model to run directly on the i.MX 93 Neural Processing Unit.

## 5.1 gPTP stack

The GenAVB/TSN stack provides a gPTP stack following the IEEE 802.1AS-2020 standard for both Endpoints and Bridges. In this case, the i.MX RT 1170 and the i.MX 93 are gPTP endpoints and the i.MX RT 1180 is a gPTP bridge.

Once the gPTP stack is launched, the targets start to send "Announce" messages to select the Grand Master following the best master clock selection algorithm (BMCA): the configuration file `/etc/genavb/fgptp.cfg` specifies if the device has Grand Master capabilities and the "standard" profile that enables the dynamic selection of the highest-quality clock as Grand Master. In our scenario, both gPTP endpoints have a clock priority1 of 248 and the gPTP bridge has a clock priority1 of 246. Therefore, the i.MX RT 1180 is selected as Grand Master and both the i.MX RT 1170 and i.MX 93 are selected as Slaves.

Following the Grand Master selection, the calculation of the link delay is computed: each device sends PathDelayReq messages and their neighbor devices reply with PathDelayResp and PathDelayRespFollowup messages. Finally, the i.MX RT 1180 sends Sync and Follow-up messages to the slaves so that they can calculate their clock offset. At each of these stages, each device timestamps packet transmission or reception, enabling the synchronization of each node's clock, and calculation of the link delay per the gPTP specification.

## 5.2 TSN application

The TSN example application provides example code and reusable middleware exercising the GenAVB/TSN API. It implements a control loop similar to industrial use cases requiring cyclic isochronous exchanges over the network. The TSN endpoints run their application synchronized to a common time grid in the same gPTP domain so that they can send and receive network traffic in a cyclic isochronous pattern. The application cycle time is equal and synchronous to the network cycle time. Currently the cycle is configured with a period of 2 ms. When the application is scheduled, frames from other endpoints are ready to be read and at the end of the application time frames are sent to other endpoints.

For this scenario, the i.MX 93 is configured as a serial controller and the i.MX RT 1170 is configured as a serial IO device. The controller sends the robot commands (or empty messages) to the IO device through the TSN network; the IO device responds with the correct information (ACK, robot position, and so on).

### 5.2.1 i.MX 93

The TSN application running on the i.MX 93 is composed of three threads: the TSN timer thread, the serial controller thread, and the statistics thread.

The role of the TSN timer thread is to schedule correctly the cyclic task corresponding to the serial controller transmission and reception of packets in the TSN network. This task scheduling is synchronous to the network scheduling thanks to the phc2sys configuration that synchronizes the system clock to the PTP clock. Additionally, this thread handles the messages coming from the robot arm and passes them to the pseudotty device handled by the serial controller thread.

The serial controller thread is the one processing the outgoing data sent through the cyclic task. To do it, the TSN application uses a pseudo-tty device (/home/root/machine-inspection/pts_file): the machine-inspection-demo python script writes the robot commands into this pseudotty device through the uARM SDK API. The serial controller thread reads this device and prepares the buffer that is sent in the cyclic task.

Finally, the statistics thread is used to print statistics on application scheduling and processing timing as well as network traffic correctness and latency.

### 5.2.2 i.MX RT 1170

In case i.MX RT 1170 is used, the TSN application is a FreeRTOS application built on top of the MCUXpresso SDK. Therefore, this application is in charge of configuring and starting both gPTP and GenAVB stacks.

Similarly to the TSN application running on i.MX 93, the i.MX-RT1170 TSN application has a cyclic task that handles the transmission and reception of packets in the TSN network. This thread also processes the received data that contains the robot commands from the i.MX 93 and sends these commands to the robot arm through the UART. The IO device’s cyclic task is started with a half cycle offset from the 2 ms period to reduce the processing latency.

The serial IO device thread processes the robot’s answers and prepares the buffer to be sent in the cyclic task.

## 5.3 Quality of service

The PTP algorithm implemented in each node ensures that the system clock and the gPTP clock are synchronous. It means that the TSN application can schedule with a greater precision the transmission of packets in the network. Additionally, the packets used for this demonstration use the VLAN tag to specify the transmission queue to be used: the VLAN priority of the transmitted packets is 5 that forces the usage of queue 2. It allows using different queues for best effort traffic and the TSN application traffic.

Finally, the Qbv configuration ensures that each queue is served during each 2 ms period:

```
tc qdisc add dev eth1 parent root handle 100 taprio \
num_tc 3 \
map 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 \
queues 1@0 1@1 1@2 \
base-time 000500000 \
sched-entry S 0x2 400000 \
sched-entry S 0x5 1600000 \
flags 0x2
```

Queue 2 is opened while queues 0 and 1 are closed for a 400 us period before queue 2 is closed and queues 0 and 1 are opened for the remaining 1600 us of the 2 ms cycle time.

In conclusion, the 400 us time window is reserved and guaranteed for controlling traffic alone. Best-effort traffic cannot use this time period. Since the opening (and closing) time of these time windows is synchronized across all nodes in the network, 400 us is enough time for the 160 Bytes long robot command to be transmitted on a 1 Gbit/s link.

### 5.4 eIQ software

eIQ ML software provides the support of different inference engines and optimized libraries. In this scenario, the Python script uses the TensorFlow Lite implementation for the i.MX 93. The interpreter loads and runs a MobileNet-SSD neural network model optimized specifically for the i.MX 93 into the i.MX 93 Neural Processing Unit (NPU), which allows the detection of objects captured through the AP1302 camera.

## 6 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

## 7 Revision history

**Table 1. Revision history**

| Document ID | Release date | Description |
|---|---|---|
| AN14287 v.1.0 | 22 April 2024 | Initial version |

# Legal information

## Contents

**Date of release: 22 April 2024**
**Document identifier: AN14287**