B. TECH. PROJECT REPORT

On

Condition Based Monitoring Device

BY
Jashandeep Singh Bhullar (150002019)

DISCIPLINE OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2018

Condition Based Monitoring Device

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degrees

of
BACHELOR OF TECHNOLOGY
in
ELECTRICAL ENGINEERING

Submitted by:
Jashandeep Singh Bhullar

Guided by:
Dr. Srivathsan Vasudevan,
Assistant Professor,
Electrical Engineering,
Indian Institute of Technology Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2018

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **"Condition Based Monitoring Device"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in **Electrical Engineering** completed under the supervision of **Dr. Srivathsan Vasudevan, Assistant Professor, Electrical Engineering, IIT Indore** is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Signature and name of the student with date

CERTIFICATE by BTP Guide

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Signature of BTP Guide with date and designation

Preface

This report on "**Condition Based Monitoring Device**" is prepared under the guidance of **Dr. Srivathsan Vasudevan**, Assistant Professor, Electrical Engineering, IIT Indore.

This work is the result of my internship program at Analog Devices, Banglore. Throughout this report, a detailed description of the technologies that have been used to design and implement a condition based monitoring device which can analyze and classify extracted signatures sent to a central server. The implementation is tested for its different inputs and results are presented in a clear and concise manner. I have tried to the best of my ability and knowledge to explain the content in a lucid manner. We have also added figures to make it more illustrative.

Acknowledgments

I would like to thank my B.Tech Project supervisor **Dr. Srivathsan Vasudevan** for his constant support in structuring the project and for his valuable feedback throughout the course of this project. He gave me an opportunity to discover and work in such an interesting domain. His guidance proved really valuable in all the difficulties we faced in the course of this project.

I am really grateful to **Mr. Pravinkumar Angolkar**, who is the Strategic Business Development Manager at Analog Devices. He also provided valuable guidance and helped with the problems while working on various technologies. He provided the initial pathway for starting the project in the right manner and provided useful directions to proceed along whenever necessary.

We are also thankful to all our family members, friends, and colleagues who were a constant source of motivation. We are really grateful to Dept. of Electrical Engineering, IIT Indore for providing with the necessary hardware utilities to complete the project. We offer sincere thanks to everyone who else knowingly or unknowingly helped us complete this project.

Jashandeep Singh Bhullar

1500002019

The Discipline of Electrical Engineering

Indian Institute of Technology Indore

Abstract

The sensing platform capable of running the Otosense signature extraction algorithm on networked edge nodes and sending the signatures to a central server for analysis and classification. The edge nodes can take multiple streams of data including acoustic and vibration information along with industrial command instructions if available. The platform is designed to be modular with several options being included if the end customer requires them as well as supporting external sensor modules.

In this project, we have used the Variscite Board VAR-SOM-MX6. Yocto project is a framework for creating a Linux distribution for embedded devices. Its layering mechanism makes it easy to add Linux to new target devices highly customized for a particular platform; it can include custom startup scripts, software packages built with a high degree of optimization for a particular architecture, and different user interfaces from full Gnome desktop to a simple a serial console. We have presented Time Synchronisation Task for the audio input. Instead of using the Real Time Clock (RTC) we have used the Network Time Protocol (NTP) we are widely used for getting the correct time at any time. Along with this, we have also introduced EPIT Timer which when used as a kernel driver module adds more functionality to the system.

Also in the project is included SCiO which is based on NIR (near infrared) spectroscopy where for a given molecule, the normal mode of vibration corresponds to internal atomic motions in which all atoms move in phase with the same frequency but with different amplitude.

Table of contents

List of Figures

List of Tables

Chapter 1: Introduction

This chapter highlights the background and motivation for the project. The problem statement has been described for the project. Towards the end, the objectives are briefly outlined and the future scope is also discussed.

1.1 Background

Embedded systems are programmed controlling and operating systems with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. Ninety-eight percent of all microprocessors are manufactured to serve as an embedded system component.

Examples of properties of typically embedded computers, when compared with general-purpose counterparts, are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with. However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functions, well beyond those available. For example, intelligent techniques can be designed to manage power consumption of embedded systems.

Modern embedded systems are often based on microcontrollers (i.e. CPUs with integrated memory or peripheral interfaces),[7] but ordinary microprocessors (using external chips for memory and peripheral

interface circuits) are also common, especially in more complex systems. In either case, the processor(s) used may be ranging from general purpose to those specialized in a certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor.

1.2 About the company

Analog Devices is an American multinational semiconductor company. Analog Devices is headquartered in Norwood, Massachusetts, with regional headquarters located in Shanghai, China; Munich, Germany; Limerick, Ireland; and Tokyo, Japan. Analog Devices has fabrication plants located in the United States and in Ireland. Headquarters in India are located in Bangalore.

Analog Devices is a global leader in the design and manufacturing of analog, mixed-signal, and DSP integrated circuits to help solve the toughest engineering challenges. Its products are ranging from Amplifiers, Audio & Video Products to Power Management, Processors, DSP, Sensors & MEMS and more.

ADI has been innovating at the intersection of the digital and real world for decades. The Analog Garage is here to help you turn your disruptive ideas into tomorrow's realities. We provide entrepreneurs with a path to propose, explore and scale new technologies and new business models. We partner, mentor, and finance entrepreneurs with ideas that solve hard problems in the real world; the noisy, the messy, the difficult: the analog world.

1.3 Motivation of the work

Embedded systems are commonly found in consumer, industrial, automotive, medical, commercial and military applications. Telecommunications systems employ numerous embedded systems from telephone switches for the network to cell phones at the end user. Computer networking uses dedicated routers and

network bridges to route data. The application-specific architecture of these systems enables us to create solutions to any problem statement at hand.

Chapter 2: About VAR-SOM-MX6

2.1 Specifications of VAR-SOM-MX6

The VAR-SOM-MX6 is a high-performance System-on-Module. It provides an ideal building block that easily integrates with a wide range of target markets requiring rich multimedia functionality, powerful graphics, and video capabilities, as well as high-processing power. Compact, cost-effective and with low power consumption, VAR-SOM-MX6 secures an Intel Atom performance level.

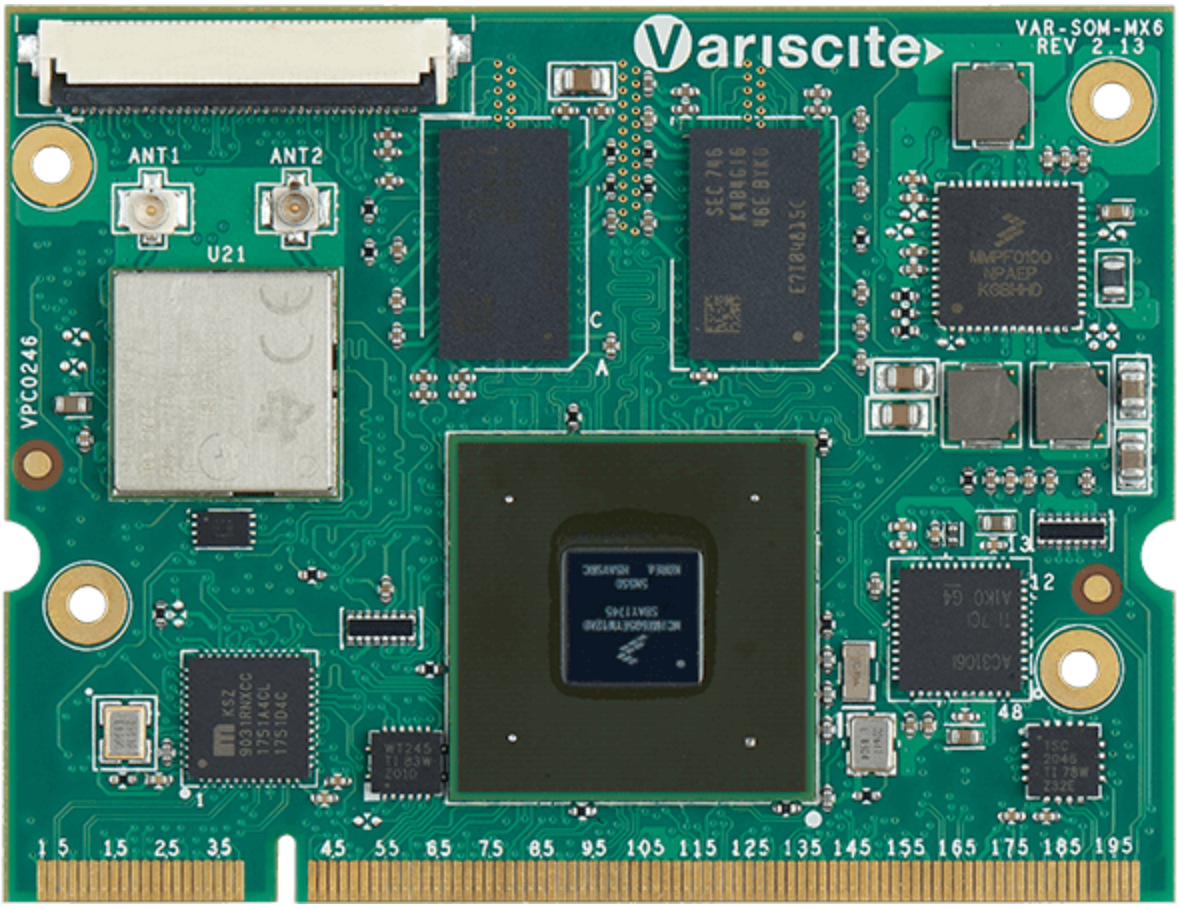

Fig. 2.1 VAR SOM MX6

Feature Summary:

- Freescale i.MX6 series SoC (Single/Dual /Quad ARM® Cortex™-A9 Core, 1.2 GHz)
- Up to 16 Gb DDR3 RAM
- 8Gb NAND Flash for storage memory/boot
- 2 x LVDS display interface
- HDMI V1.4 interface
- 1 x MIPI DSI
- Touch panel interface
- Parallel & serial camera interface
- Onboard 10/100/1000 Mbps Ethernet PHY
- WLAN (802.11 b/g/n) / BT
- 1 x USB 2.0 host, 1 x OTG
- 2 x SD/MMC
- Serial interfaces (SPI, I2C, UART, I2S,)
- CAN Bus
- Stereo line-In / headphones out
- Digital microphone
- Single 3.3 V power supply
- 67mm x 51mm, 200 pins SO-DIMM Connector

2.2 Block Diagram

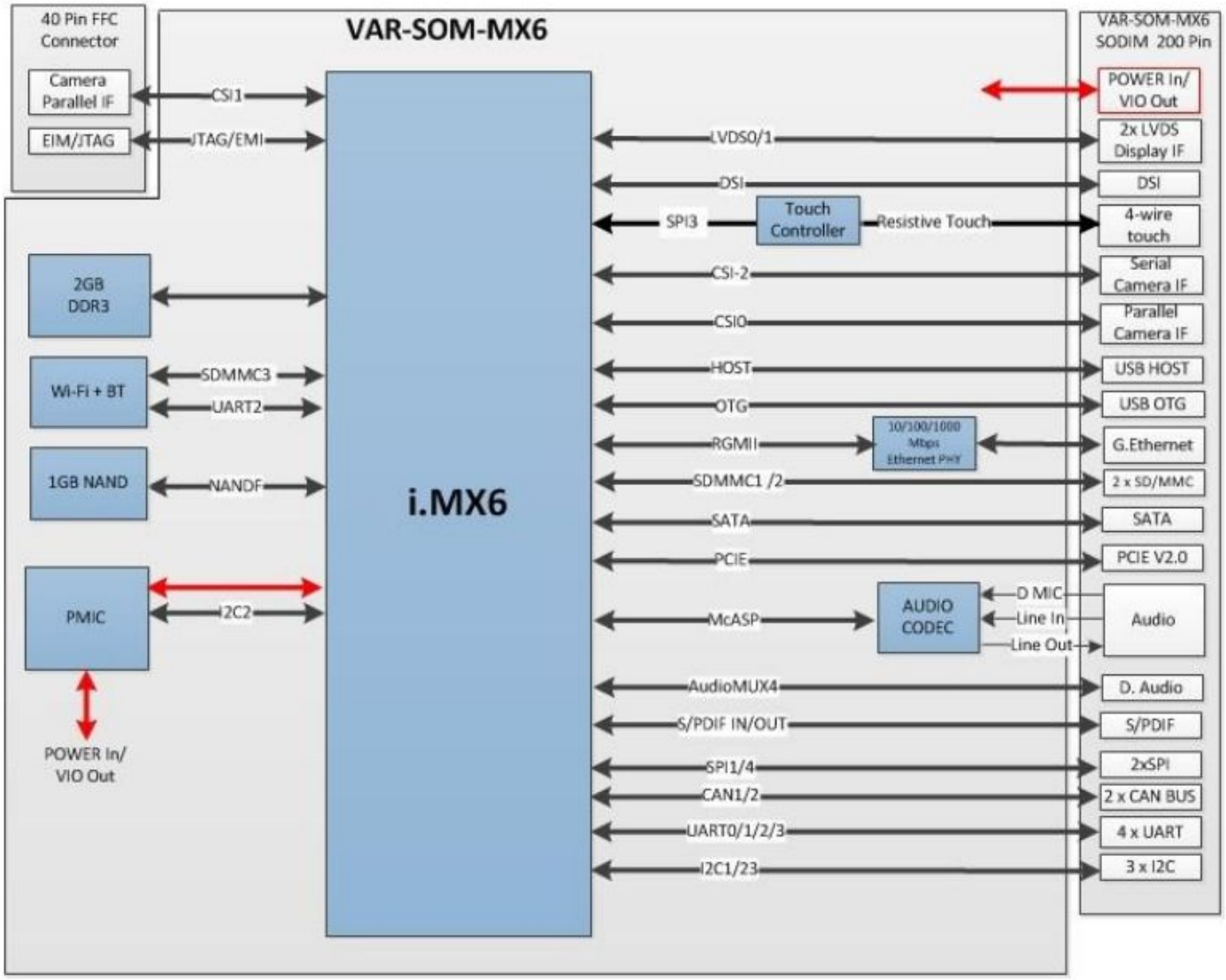

Fig. 2.2 VAR SOM MX6 Block Diagram

2.3 Variscite Development workflow

2.3.1 Overview

The VAR-SOM-MX6 is a high-performance System-on-Module. It provides an ideal building block that easily integrates with a wide range of target markets requiring rich multimedia functionality, powerful graphics, and video capabilities, as well as high-processing power. Compact, cost-effective and with low power consumption, VAR-SOM-MX6 secures an Intel Atom performance level.

2.3.2 Setting up the Yocto-Linux build environment

Download the Yocto Rocko based on Freescale BSP

Step 1: Install the required packages

```
$ sudo apt-get update
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping libsdl1.2-dev xterm
$ sudo apt-get install autoconf libtool libglib2.0-dev libarchive-dev python-git \
sed cvs subversion coreutils texi2html docbook-utils python-pysqlite2 \
```

help2man make gcc g++ desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev \

mercurial automake groff curl lzop asciidoc u-boot-tools dos2unix mtd-utils pv \

libncurses5 libncurses5-dev libncursesw5-dev libelf-dev zlib1g-dev

Step 2: Download the Rocko Yocto BSP source files

1. *Set the identity*

$ git config --global user. name "Your Name"

$ git config --global user.email "Your Email"

2. *Download the Repo utility and set the permissions*

$ mkdir ~/bin

$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo

$ chmod a+x ~/bin/repo

$ export PATH=~/bin:$PATH

Note: ~/bin directory creation step may not be needed if the bin folder already exists

3. Create the directory to hold the downloaded repo files

$ mkdir ~/var-fslc-yocto

$ cd ~/var-fslc-yocto

4. Download the Rocko Yocto BSP repo files

$ repo init -u https://github.com/varigit/variscite-bsp-platform.git -b refs/tags/rocko-fslc-4.9.11-mx6-v1.2

$ repo sync -j4

Baseline BSP release: rocko-fslc-4.9.11-mx6-v1.2

Linux Kernel Version: 4.9.11

Setup configuration and Build Yocto

Step 1: Create the build environment

Set MACHINE=<machine name> DISTRO=<distro name> <build directory>

$ cd ~/ var-fslc-yocto

$ MACHINE=var-som-mx6 DISTRO=fslc-framebuffer . setup-environment build_fb

Step 2: Launch bitbake

$ bitbake fsl-image-gui

The bitbake take several hours to complete for the first time. Please make sure the internet connection till completion of the process. The subsequent bitbake uses cached data hence will be faster.

The resulted images are located in the build directory at *tmp/deploy/images/var-som-mx6.*

Note:

- fslc-framebuffer: the Distro for Framebuffer graphical backend. This distro doesn't include X11 and Wayland features.
- fsl-image-gui: the default Variscite demo image with GUI and without any Qt5 content. This image recipe works on all backends for X11, Frame Buffer and Wayland
- bitbake is failing for the distros: fslc-x11, fslc-xwayland

2.3.3 Create Bootable SD card

The recommended size of SD card is minimum 8GB; (though 4GB SD card can be sufficient).

Using prebuilt images

Step 1: Download rocko-fslc-4.9.11-mx6-v1.2.img.gz from Variscite's FTP server: ftp://customerv:Variscite1@ftp.variscite.com/VAR-SOM-MX6/Software/fslc

Step 2: Flash the image to SD card

1. Using Ubuntu (linux PC)

$ cd ~/Download

$ sudo umount /dev/sdX*

$ zcat rocko-fslc-4.9.11-mx6-v1.2.img.gz | sudo dd of=/dev/sdX bs=1M && sync

Note:

- ~/Download is the directory to which the rocko-fslc-4.9.11-mx6-v1.2.img.gz file is downloaded.
- Replace /dev/sdX with your actual device. Eg /dev/sdb

2. Using a Windows-based host

- Download Win32 Disk Imager from https://sourceforge.net/projects/win32diskimager/ and install it.
- Extract downloaded rocko-fslc-4.9.11-mx6-v1.2.img.gz file to get <image name>.img (using 7-Zip for example)
- Insert your SD card into your PC.
- No need to format the SD card before writing the image to it, as the card will be formatted once it will be flashed.
- Run the file named Win32DiskImager.exe (in Windows 7,8 and 10 we recommend that you right-click this file and choose “Run as administrator”).
- If the SD card (Device) you are using isn’t found automatically. Click on the drop down box and select it
- In the Image File box, choose the <image name>.img file you have extracted previously
- Click Write
- After a few minutes, you receive a notification that your SD has been created successfully.

3. Using a Mac OS X host

Use the dd utility for flashing the images.

Refer steps provided at:
https://www.thefanclub.co.za/how-to/dd-utility-write-and-backup-operating-system-img-files-memory-card-mac-os-x#osx

Using the images built in bitbake

The bitbake generates images at //build_fb/tmp/deploy/images/var-som-mx6

Insert the SD card to Ubuntu Linux PC and find the name of the partition by giving the command:

$ ls /dev/sd*

To flash the image to SD card run give the below command:

$ sudo dd if=tmp/deploy/images/var-som-mx6/fsl-image-gui-var-som-mx6.sdcard of=/dev/sdX bs=1M && sync

2.3.4 Booting VAR SOM from SD Card

Make the necessary connection VAR SOM MX6 board

- Insert the SD card
- Connect the RJ45 Ethernet Jack
- Connect the FTDI (Note: Pin 1 of the FTDI connector should be connected Pin 2 of the FTDI head on the board)
- Connect the RS232 D9 connector to COM port of the PC
- Connect the power cord, ensure power switch is in OFF position

For console prints on

1. Windows PC use Putty –
2. Ubuntu PC

Install the picocom package on Ubuntu PC

$ sudo apt-get install picocom

Launch the picocom terminal

$ picocom -b 115200 /dev/ttyS0

To boot from SD card- press and hold the SW3 (boot selection) switch, switch the power ON, leave the SW3 once prints start appearing on the console terminal.

Type ‘root’ to login.

Note: The VAR SOM MX6 may fail to mount the rootfs. In such scenario increase the size of the rootfs partition of the SD card (generally /dev/sdx2).

Use GParted utility tool on Ubuntu PC.

2.3.5 Build and Install toolchain and SDK

The Yocto toolchain and SDK has to be built and installed on Ubuntu development PC. Yocto Linux build environment has to be set prior to build and install the toolchain and SDK.

Build the toolchain

Run the below commands to build toolchain

$ cd ~/var-fslc-yocto

$ MACHINE=var-som-mx6 DISTRO=fslc-framebuffer . setup-environment build_fb

$ bitbake meta-ide-support

$ bitbake meta-toolchain

Build the SDK

Run the below commands to build the SDK

$ bitbake -c populate_sdk fsl-image-gui

Install the toolchain and SDK

Install the toolchain and SDK by executing the .sh file generated at /tmp/deploy/sdk. This will install the complete SDK at /opt/

$ tmp/deploy/sdk/fslc-framebuffer-glibc-x86_64-fsl-image-gui-armv7at2hf-neon-toolchain-2.4.3.sh

The toolchain and SDK will be installed in the directory /opt/fslc-framebuffer/2.4.3

2.3.6 Building a sample application program

Create a simple Hello World C program in ~ directory; eg. hello.c

Source environmental set up by running

$ source /opt/fslc-x11/2.4.3/environment-setup-armv7at2hf-neon-fslc-linux-gnueabi

Build the hello.c file by running below command.

$ $CC hello.c –o hello

This creates the bin file ‘hello’

2.3.6 Executing the application program

Boot the VAR SOM Board using SD card. Ensure the board is connected to the network

Copy the ‘hello’ file to target board using scp command.

$ scp hello root@<ip addr>:/home

Execute the ‘hello’ program from VAR SOM Board console.

$ cd /home

$ ./hello

Chapter 3: Condition Based Monitoring Device

3.1 Overview

It is a sensing platform capable of running the Otosense signature extraction algorithm on networked edge nodes and sending the signatures to a central server for analysis and classification. The edge nodes can take multiple streams of data including acoustic and vibration information along with industrial command instructions if available. The platform is designed to be modular with several options being included in case the end customer requires them as well as supporting external sensor modules.

3.2 Time Synchronization

The audio data that is received, needs to be time-stamped periodically for the learning algorithms to work. When the system boots, the hardware clock, that is, Real Time Clock is synchronized with the time obtained by the network.

To implement this system, we need to use NTP, Network Time Protocol. This enables us to obtain the correct time reading through the network, no matter where your device is situated.

3.2.1 Limitations of Real Time Clock

Time usually just advances. If you have communicating programs running on different computers, time still should even advance if you switch from one computer to another. Obviously, if one system is ahead of the

others, the others are behind that particular one. From the perspective of an external observer, switching between these systems would cause time to jump forward and back, a non-desirable effect.

As a consequence, isolated networks may run their own wrong time, but as soon as you connect to the Internet, effects will be visible.

Even on a single computer, some applications have trouble when the time jumps backward. For example, database systems using transactions and crash recovery like to know the time of the last good state. Therefore, air traffic control was one of the first applications for NTP.

3.2.2 Network Time Protocol

The Network Time Protocol is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks.

Network Time Protocol (NTP) is widely used in order to synchronize a computer to Internet time servers or other sources, such as a radio or satellite receivers or telephone modem services. It provides accuracies typically less than a millisecond on LANs and up to a few milliseconds on WANs. Typical NTP configurations utilize multiple redundant servers and diverse network paths in order to achieve high accuracy and reliability.

It can maintain time over the public Internet to within 10 milliseconds and can perform even better over LANs. NTP time servers work within the TCP/IP suite and rely on User Datagram Protocol (UDP) port 123.

3.2.3 Features of NTP

- NTP needs some reference clock that defines the true time to operate. All clocks are set towards that true time. It will not just make all systems agree on some time but will make them agree upon the true time as defined by some standard.
- NTP uses UTC as the reference time.

- NTP is a fault-tolerant protocol that will automatically select the best of several available time sources to synchronize to. Multiple candidates can be combined to minimize the accumulated error. Temporarily or permanently insane time sources will be detected and avoided.
- NTP is highly scalable: A synchronization network may consist of several reference clocks. Each node of such a network can exchange time information either bidirectional or unidirectional. Propagating time from one node to another forms a hierarchical graph with reference clocks at the top.
- Having available several time sources, NTP can select the best candidates to build its estimate of the current time. The protocol is highly accurate, using a resolution of less than a nanosecond (about 2^-32 seconds).
- Even when a network connection is temporarily unavailable, NTP can use measurements from the past to estimate current time and error.

3.2.4 Implementation

We start by setting up an NTP client for the host computer where our operation is running. In the next step, a UDP socket is made for the communication between host computer and the NTP server.

NTP servers are located all around the world, for instance, uk.pool.ntp.org, us.pool.ntp.org etc. But the best way to choose a server is by selecting the nearest server. This is a universal URL which always returns the nearest server and that is pool.ntp.org.

Finally, if the above steps go correctly, we obtain a packet which contains seconds and fraction of seconds. This actually is in the format of UNIX Epoch time, that is, the number of seconds passed since 1970.

3.3 Timers in Variscite Board

The Variscite board houses lots of hardware timers such as General Purpose Timer (GPT), Enhanced Periodic Interrupt Timer (EPIT), Watchdog Timer (WDOG) and more.

Most of these are reserved by the system itself, expect for EPIT. So when its time to be coming up with a counter for creating custom solutions we choose EPIT over other timers.

Why we need a timer? The reason is directly related to RTC (real time clock). The RTC integrated into the device can store a time unit to a minimum of 1 second. So we have to find a way to store even the fractions of second if we have to. That's why we are using a hardware timer.

3.3.1 EPIT Timer and its features

EPIT is a 32-bit set-and-forget timer that is capable of providing precise interrupts at regular intervals with minimal processor intervention. EPIT begins counting after it is enabled by software.

EPIT has the following key features:

- 32-bit down counter with clock source selection
- 12-bit prescaler for the division of input clock frequency
- The counter value that can be programmed on the fly
- Can be programmed to be active during low-power and debug modes
- Interrupt generation when the counter reaches the compare value

3.3.2 EPIT block diagram

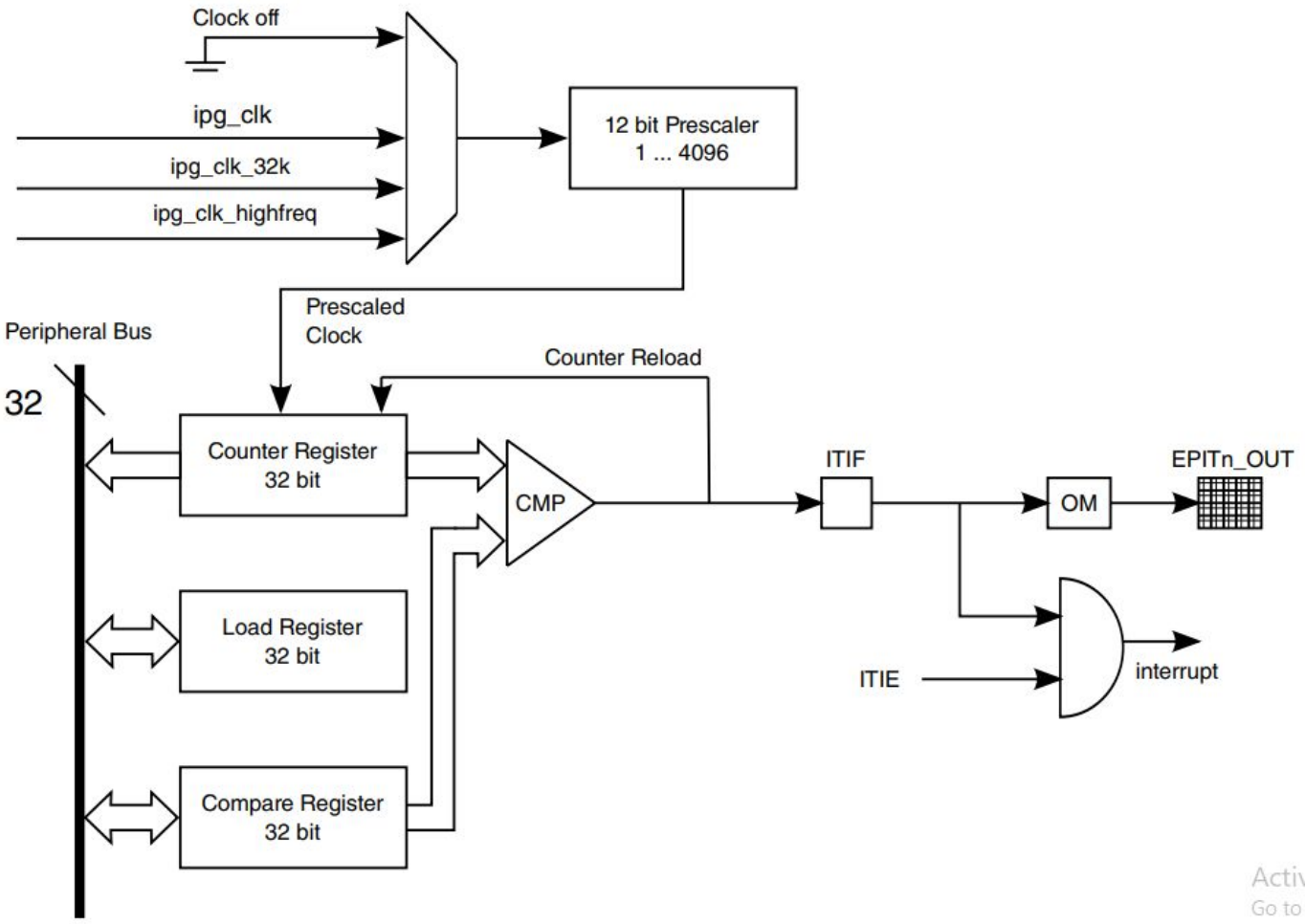

Fig 3.1 EPIT Timer Block Diagram

3.3.3 Modes of Operation

EPIT can operate in either set-and-forget or free-running mode. Use EPIT_CR[RLD] to select the desired mode.

Operating in the set-and-forget mode

To select this mode of operation, set the RLD bit in the control register (EPIT_CR).In this mode, the counter obtains its data from the load register (EPIT_LR); it cannot be written to directly from the block data bus. Whenever the counter reaches zero, the value in EPIT_LR is loaded into the counter. This value is then decremented to zero. To directly initialize the counter instead of waiting for the count to reach zero, set the EPIT counter-overwrite enable bit (EPIT_CR[IOVW]) and write to EPIT_LR with the required initialization value.

Operating in free-running mode

To select this mode of operation, clear the RLD bit. In this mode, the counter rolls over from 0000 0000h to FFFF FFFFh without reloading from the modulus register. After rolling over, the counter continues counting down. To directly initialize the counter, set the EPIT counter-overwrite enable bit (EPIT_EPITCL[IOVW]) and write to EPIT_EPITLR with the required initialization value.

3.3.4 Clock Sources

The table found here describes the clock sources for EPIT:

Clock name	Clock Root	Description
ipg_clk	ipg_clk_root	Peripheral clock
ipg_clk_32k	ckil_sync_clk_root	Low-frequency reference clock (32 kHz)
ipg_clk_highfreq	perclk_clk_root	High-frequency reference clock
ipg_clk_s	ipg_clk_root	Peripheral access clock

Table 3.1 EPIT clocks

The clock that feeds the prescaler can be selected from among the following sources:

- **High-frequency reference clock (ipg_clk_highfreq)**

This clock is provided by the Clock Control Module (CCM). This clock remains on during low-power mode when the peripheral clock is turned off, allowing EPIT to use this clock in low-power mode. In normal mode, the CCM synchronizes this clock to ahb_clk; in low-power mode, CCM switches to an unsynchronized version.

- **Low-frequency reference clock (ipg_clk_32k)**

This 32 kHz reference clock is provided by the CCM. This clock remains on in low power mode when the peripheral clock is turned off, so EPIT can use this clock during low-power mode. In normal mode, the CCM synchronizes this clock to ahb_clk; in low-power mode, CCM switches to an unsynchronized version. This

clock is derived from the external 32 kHz crystal.

- **Peripheral clock (ipg_clk)**

This is the peripheral clock (PER Clock) which is provided (and optionally gated) by the CCM. This clock is typically used in normal operations. In low-power modes, if the EPIT is programmed to be disabled (via STOPEN or WAITEN), then the peripheral clock can be switched off.

The clock input source is determined by the CLKSRC field in the control register. The clock input to the pre-scaler can also be disabled by setting CLKSRC to 0b00. This field value should only be changed after first disabling the EPIT by clearing the EN bit in the EPIT_EPITCR.

3.3.5 Operations

EPIT has a single 32-bit down counter, which starts counting when the block is enabled by software.

The start value of the counter is loaded from the EPIT load register, which can be written to at any time by the processor. The value in the compare register determines the time that the interrupt occurs.

When EPIT is disabled (EN = 0), both the main counter and the pre-scaler counter freeze their count at their current count values. When EPIT is re-enabled (EN = 1), the ENMOD bit, which is an RW bit, decides the counter value:

- If ENMOD is set, the main counter is loaded with the load value (If RLD = 1)/FFFF FFFFh (If RLD = 0) and the pre-scaler counter is reset (000h).
- If ENMOD is cleared, both main counter and pre-scaler counter restart counting from their frozen values.

If EPIT is programmed to be disabled in a low-power mode (STOP/WAIT), both the main counter and the pre-scaler counter freeze at their current count values when EPIT enters the low-power mode. When EPIT exits the low-power mode, both the main counter and the pre-scaler counter start counting from their frozen values regardless of the ENMOD bit.

A hardware reset resets all EPIT registers to their respective reset values. There is a software reset which has the same effect on all registers except for the EN, ENMOD, STOPEN and WAITEN bits in the control register. The state of these bits is not affected by software reset. A software reset can be asserted even when the EPIT is disabled.

3.3.6 EPIT Memory Map

The EPIT includes five user-accessible 32-bit registers. The following table summarizes these registers and their addresses.

Peripheral bus writes access to the EPIT control register (EPITCR) and the EPIT load register (EPITLR) results in one cycle of wait state, while other valid peripheral bus accesses are with 0 wait state.

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
20D_0000	Control register (EPIT1_CR)	32	R/W	0000_0000h	24.6.1/1216
20D_0004	Status register (EPIT1_SR)	32	R/W	0000_0000h	24.6.2/1219
20D_0008	Load register (EPIT1_LR)	32	R/W	FFFF_FFFFh	24.6.3/1219
20D_000C	Compare register (EPIT1_CMPR)	32	R/W	0000_0000h	24.6.4/1220
20D_0010	Counter register (EPIT1_CNR)	32	R	FFFF_FFFFh	24.6.5/1220
20D_4000	Control register (EPIT2_CR)	32	R/W	0000_0000h	24.6.1/1216
20D_4004	Status register (EPIT2_SR)	32	R/W	0000_0000h	24.6.2/1219
20D_4008	Load register (EPIT2_LR)	32	R/W	FFFF_FFFFh	24.6.3/1219
20D_400C	Compare register (EPIT2_CMPR)	32	R/W	0000_0000h	24.6.4/1220
20D_4010	Counter register (EPIT2_CNR)	32	R	FFFF_FFFFh	24.6.5/1220

Table 3.2 EPIT memory map

3.3.7 Implementation

EPIT Timer needs to implement its own driver as a kernel module. Writing a device driver for Linux requires performing a combined compilation with the kernel.

At its base, a module is a specifically designed object file. When working with modules, Linux links them to its kernel by loading them to its address space. The Linux kernel was developed using the C programming language and Assembler. C implements the main part of the kernel, and Assembler implements parts that depend on the architecture.

To obtain the time unit as a fraction of seconds, we form a counter which starts from 0 and ends at the clock frequency. For example, if the clock frequency is 32KHz, then time difference between consecutive counter ticks is 1/32000 of a second.

Chapter 4: SCiO

4.1 Overview

Scan objects and materials around you and get instant information about their chemical makeup sent directly to your smartphone.

Based on NIR (near infrared) spectroscopy where for a given molecule, the normal mode of vibration corresponds to internal atomic motions in which all atoms move in phase with the same frequency but with different amplitude

- **Near Infrared Spectroscopy**

The NIR spectrum originates from radiation energy transferred to mechanical energy associated with the motion of atoms held together by chemical bonds in a molecule. Although many would approach method development in a purely empirical way, knowledge of the theory can help to look at the important wavelengths and quicker optimization of the modeling stage.

Near-infrared spectroscopy is based on molecular overtone and combination vibrations. Such transitions are forbidden by the selection rules of quantum mechanics. As a result, the molar absorptivity in the near-IR region is typically quite small. One advantage is that NIR can typically penetrate much further into a sample than mid-infrared radiation. Near-infrared spectroscopy is, therefore, not a particularly sensitive technique, but it can be very useful in probing bulk material with little or no sample preparation.

- **Applications**

Agriculture

Near-infrared spectroscopy is widely applied in agriculture for determining the quality of forages, grains, and grain products, oilseeds, coffee, tea, spices, fruits, vegetables, sugarcane, beverages, fats, and

oils, dairy products, eggs, meat, and other agricultural products. It is widely used to quantify the composition of agricultural products because it meets the criteria of being accurate, reliable, rapid, non-destructive, and inexpensive.

Materials Science

Techniques have been developed for NIR spectroscopy of microscopic sample areas for film thickness measurements, research into the optical characteristics of nanoparticles and optical coatings for the telecommunications industry.

Apart from that, it also some medical uses.

4.2 SCiO Sensor

SCiO Sensor is the world's first handheld molecular sensor! SCiO Sensor was purpose-built to be both easy and intuitive. Because your SCiO Sensor is so small and light, a large number of features are packed into a very small device. Make sure you familiarize yourself with all of the features and functions before you start scanning to avoid low-quality results. The diagram below details each functional part of your new SCiO Sensor and accessories:

Fig 4.1 SCiO Sensor and Cover

1. Molecular Sensor (takes the scan)
2. Illumination (illuminates the sample area during scan)
3. Function Button (On/Off/Scan/Calibrate)
4. Illumination Ring (SCiO Status Indicator)
5. USB Charging Port
6. Charging LED (Charge/Power Status)
7. Calibration Module (refer to Calibrate SCiO for more information)

Your SCiO Sensor shipped with a number of accessories. Additional accessories such as a liquid scanning accessory are currently in design and testing.

SCiO Optical Shade:

A special add-on device to eliminate environmental noise, excess ambient light and interference. Place the Shade over the SCiO Molecular Sensor when scanning and remove and store on the back of device (magnets will hold the device in place) when not in use.

Fig 4.2 SCiO Optical Shade

SCiO Solid Sample Holder:

In cases where your sample material is too small to fill the illumination area of the sensor, and it is a fully dry solid, the sample holder enables you to get an accurate and consistent scan of your materials. It is critically important not to put any liquids into the sample holder and to cover the holder when not in use.

Fig 4.3 SCiO Solid Sample Holder

4.3 App Development Workflow

Using the SCIO SDK that is shipped with Developer account we can build the app in the following manner:

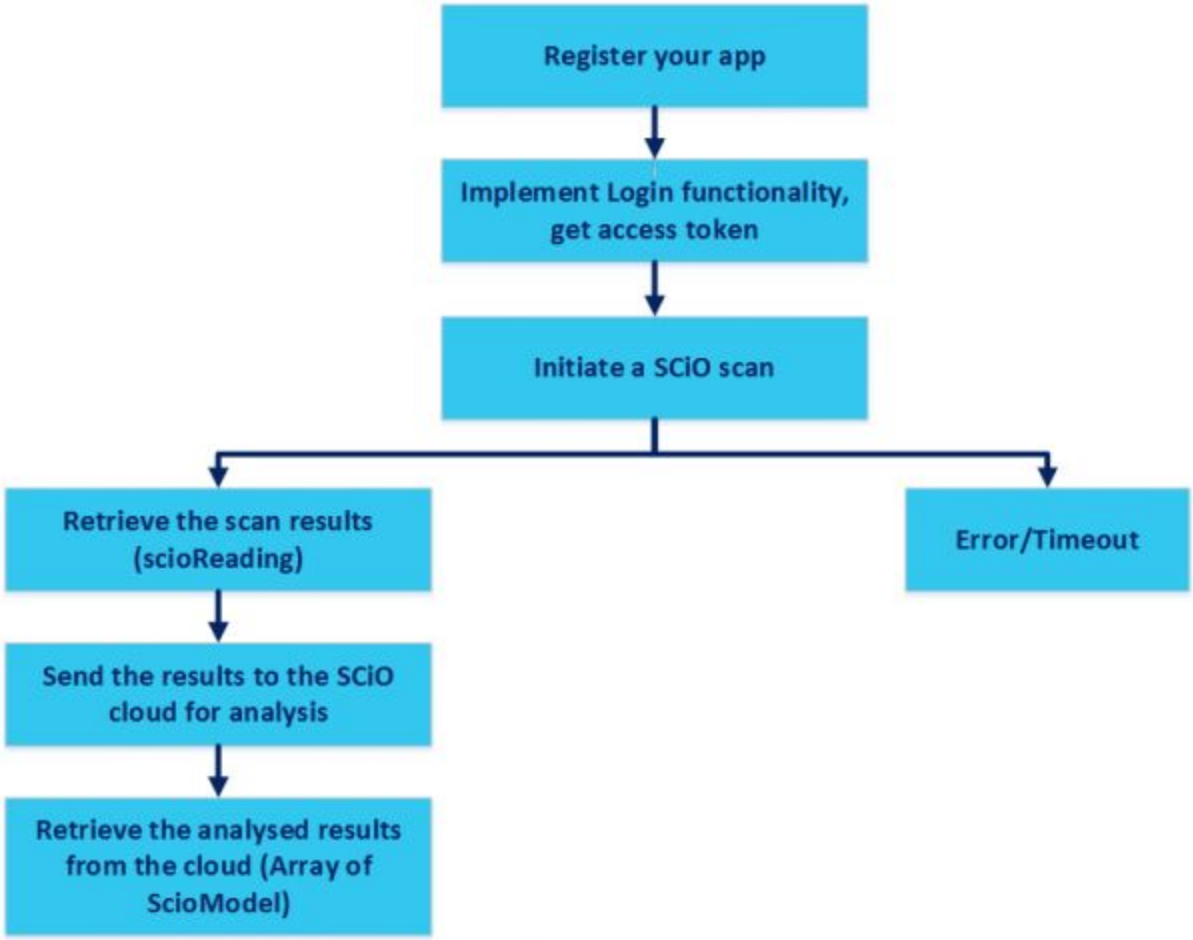

Fig 4.4 App Development Workflow

4.3.1 Register your app

Create and register your app in SCiO Lab to receive an Application ID (Key) and Application Secret.

4.3.2 Implement Login functionality

Use the SCiO login activity to allow the end user to login to the SCiO Cloud.

When using an app created with the SCiO Mobile SDK, after the end user logs in for the first time, an access token is saved on the user's mobile device. As the access token does not have an expiry date the end user will not need to log in again.

4.3.3 Connect to the SCiO sensor

Use the Bluetooth Low Energy API to ensure thathe Bluetooth settings are enabled in your Android device. Search and connect to the SCiO Sensor.

Note:

A SCiO Sensor can only connect to one mobile device at a time.

4.3.4 Initiate a SCiO scan

Call the *ScioDevice.scan* method to scan a sample. The results are returned in a ScioReading object. A scan returns one of the following results:

- *onSuccess*: When the scan is successful, the scan returns a ScioReading object with the sample details.

- *onNeedCalibrate*: Indicates that the SCiO sensor should be calibrated. The scan failed and no reading is returned.
- *onError*: Indicates that the scan failed.
- *onTimeout*: Indicates that there was a timeout.

4.3.5 Send the SCiO reading to the Cloud for analysis

Send the sample reading to the cloud together with a ModelID. The sample is compared to the model and the results returned to the mobile device. The model is defined in the ScioLab app and holds definitions for various items such as cheese or medications.

Chapter 5: Conclusion and Future Work

5.1 Conclusion

In this project, we have presented an embedded system solution which helps us analyze and classify different signature extractions that are sent to a central server. We understood the importance of Network-Based Time which is the correct always and anywhere. We also made time synchronization of incoming audio data through NTP protocol.

Along with this, we also saw how the built-in EPIT timer can add high functionality to the system. It has lots of options so that we can easily adjust to our needs.

We have presented and understood the importance of Near Infrared Spectroscopy in our daily lives to check the quality of food products in general. Also, we understood the architecture behind a basic app that works with the SCiO sensor to gather data and send it for analysis.

5.2 Future Work

SWUpdate stands for Software Update for Embedded Systems. It is a Linux Update agent with the goal to provide an efficient and safe way to update an embedded system. SWUpdate supports local and remote updates, multiple update strategies and it can be well integrated into the Yocto build system by adding the meta-swupdate layer.

It supports the common media on embedded devices such as NAND flashes, UBI volumes, SD / eMMC, and can be easily extended to introduce project specific update procedures. Pre- and post-install scripts are supported, and a LUA interpreter helps to customize the update process.

An update package is described by the sw-description file, using the libconfig syntax or JSON. It is even possible to use Lua with a custom syntax.

Here is a short list of the main features:

- Install on embedded media (eMMC, SD, NAND flash)
- Allow delivery single image for multiple devices
- Multiple interfaces for getting software
    - local storage
    - integrated web server
    - integrated REST client connector to hawkBit
    - remote server download
- Software delivered as images, gzipped tarball, etc.
- Allow custom handlers for installing FPGA firmware, microcontroller firmware via custom protocols.
- Power-Off safe
- Hardware / Software compatibility.

When completed, the program will check for update, download the image from any the interfaces described above and shift to the new image.

References

- Analog Devices Inc.: http://www.analog.com/en/index.html
- Variscite: http://variwiki.com/index.php?title=Main_Page
- VAR SOM MX6: http://variwiki.com/index.php?title=VAR-SOM-MX6
- Intro (tutorial) : http://variwiki.com/index.php?title=Yocto_Start_Here&release=RELEASE_ROCKO_V1.2_VAR-SOM-MX6
- NTP: https://en.wikipedia.org/wiki/Network_Time_Protocol
- Repo sync and Build environment setup http://variwiki.com/index.php?title=Yocto_Build_Release&release=RELEASE_ROCKO_V1.2_VAR-SOM-MX6
- Creating Bootable SD card http://variwiki.com/index.php?title=Yocto_Recovery_SD_card&release=RELEASE_ROCKO_V1.2_VAR-SOM-MX6
- Yocto quick start guide https://www.yoctoproject.org/docs/2.4.2/yocto-project-qs/yocto-project-qs.html