



S32G Linux PFE Driver User's Manual

Contents

1	Revision History	3
2	Introduction	4
3	Building Procedure	5
3.1	Building the driver	5
3.1.1	Variant 1: Building within NXP Auto Linux BSP source tree by Yocto	5
3.1.2	Variant 2: Building standalone	5
4	Usage	7
4.1	Prerequisites	7
4.1.1	NXP S32G274A silicon platform EVB, RDB or RDB2	7
4.2	Supported development boards	7
4.3	Running the driver	7
4.4	The driver configuration - device tree	8
4.5	Performance consideration	11
4.6	Silicon cut 1.1 backward compatibility	11
5	Master-Slave feature	12
5.1	Prerequisite	12
5.2	Master	12
5.3	Slave	12
5.4	Running	13
6	IEEE1588 Support	15
6.1	Clock configuration	15
6.1.1	Internal Timestamp Mode(full line figure 6.1)	16
6.1.2	External Timestamp Mode(dashed line figure 6.1)	16
6.2	Driver interface	16
6.2.1	IOCTL	16
6.2.2	PTP hardware clock device	17
6.2.3	ethtool	17
6.3	BSP yocto support	17
7	LibFCI	18
7.1	The libfci	18
7.2	Building libfci library	18
7.3	Entering Yocto development environment	18

Chapter 1

Revision History

Revision	Change Description
preEAR 0.4.0	Initial version for preEAR (FPGA/x86 platform) [JPet]
preEAR 0.4.1	Added VDK info [JPet]
preEAR 0.4.3	Removed VDK, Added S32G [JPet]
EAR 0.8.0	Added device-tree config [JPet]
EAR 0.8.0 Patch 1	Added libfci [JPet]
BETA 0.9.0	DTS update for kernel 5.4-rt [JPet]
BETA 0.9.1	Added Master/Slave [JPet]
BETA 0.9.2	Added performance info [JPet]
BETA 0.9.3	Added PTP [OSpac], Added cut1.1 info [JPet]

Table 1.1: Revision History

Chapter 2

Introduction

The Linux PFE driver is, in general, an OS-specific SW component responsible for the complex PFE management. It consists of three main functional blocks:

- The platform driver
This part covers the initial, low-level PFE HW bring-up, configuration, firmware upload, and the SW representation of the PFE HW components. This part can be described as a low-level PFE driver providing the interface to the hardware (including the firmware).
- The data-path driver
Data-path driver covers tasks related to the Ethernet data traffic in terms of passing the packets between the PFE and the networking stack. This includes the implementation of the Host Interface (HIF) driver and connection with the host OS-provided the networking stack.
- Control path driver
The block in charge of the functionality related to runtime PFE engine configuration and monitoring, implementing the Fast Control Interface (FCI) endpoint which is accessible from the user-space through related FCI library.

The driver runs within the target host OS environment and connects the PFE with networking stack. It provides access to physical Ethernet interfaces via exposing logical interfaces to the OS, and the OS together with user's applications can use the Ethernet connectivity via standard OS-provided interfaces (i.e. network sockets).

Chapter 3

Building Procedure

3.1 Building the driver

The PFE driver consists of number of smaller software modules. The main module producing the final driver is called *linux-pfeng* and depends on all the others. Successful build gives the final driver library *pfeng.ko* which is an Ethernet driver for particular Linux kernel. Its location within the project tree in `sw/linux-pfeng/`. There are two ways to build the driver:

1. Integrated build by NXP Automotive Linux BSP, powered by Yocto
2. Standalone

3.1.1 Variant 1: Building within NXP Auto Linux BSP source tree by Yocto

1. As prerequisite, save the PFE firmware files to any location on the local disk. Ask NXP representative for PFE firmware package.
2. Build standard NXP Auto Linux BSP to check that all necessary components are ready. This step is not only for verification but it precompiles most of necessary part of BSP which will be used later for creating sdcard image with included pfeng Linux driver.
3. Configure additional component by adding the following two lines to the `conf/local.conf`

```
DISTRO_FEATURES_append = " pfe "  
PFE_LOCAL_FIRMWARE_DIR = "<directory path where fw file was saved>"  
or (on ALB BSP 28.0 or higher)  
NXP_FIRMWARE_LOCAL_DIR = <directory path where fw file was saved>"
```
4. Rebuild BSP to get PFE driver + firmware included in the sdcard image

3.1.2 Variant 2: Building standalone

1. As prerequisite check all necessary development requirements:
 - (a) Host development GNU toolchain, including GNU-cc, GNU-make
 - (b) Linux kernel development files
2. Go to `sw/linux-pfeng/`.

3. Make sure that the following environment variables are set and points to the right directories:

- (a) KERNELDIR

The directory where the Linux kernel development files are located.

- (b) Optionally also PLATFORM

The name of the GNU toolchain platform. In case of NXP Auto Linux BSP, it is "aarch64-linux-gnu"

4. Clean working directories:

```
make KERNELDIR=<path to kernel> PLATFORM=aarch64-linux-gnu drv-clean
```

5. Build the driver:

```
make KERNELDIR=<path to kernel> PLATFORM=aarch64-linux-gnu all
```

Chapter 4


Usage

4.1 Prerequisites

4.1.1 NXP S32G274A silicon platform EVB, RDB or RDB2

For usage the software on S32G EVB or RDB/RDB2 boards, the following shall be fulfilled:

- Compatible PFE firmware files.

 *The compatibility requirements can be find in the driver public repository:
<https://source.codeaurora.org/external/autobsp32/extra/pfeng>*

- Auto Linux BSP version 27.0 or higher for creation of sdcard bootable image

 *Note that PFE firmware is being delivered as a standalone product package.*

4.2 Supported development boards

NXP offers two development boards for S32G274A:

1. S32G-VNP-EVB
2. S32G-VNP-RDB2

Please refer to board specific user guide to check possible ethernet connectivity. Also check 'NXP Automotive Linux BSP User Manual' which provides comprehensive information for ethernet controllers, including PFE, with supported configurations for U-boot and Linux.

4.3 Running the driver

1. The *pfeng.ko* driver is embedded in the Auto Linux BSP sdcard image and as such is automatically loaded on Linux startup.
2. To check available network interfaces:

```
root@s32g274aevb:~# ifconfig -a
```

there should be some, based on config, *pfeX* interfaces.

3. Configure the IP addresses and bring the interfaces up by executing:

```
root@s32g274aevb:~# ifconfig pfe<0-2> <interface_ip_address>/<mask>
```

4. The *ping* utility can be used to test the connection:

```
root@s32g274aevb:~# ping <remote_ip_address>
```

4.4 The driver configuration - device tree

The driver is configurable by changes in the device-tree pfeng node. Usually, the device tree files are using multi-level organization, for example the *fsl-s32g274a-evb.dtb* is built by integration:

1. Family DT file *fsl-s32-gen1.dtsi*
2. SoC DT file *fsl-s32g274a.dtsi*
3. Board specific DT file *fsl-s32g274a-evb.dts*

Device tree bindings for fsl-pfeng:

```
*
* Copyright 2020-2021 NXP
*
* NXP S32G274A PFE networking accelerator (pfeng)
*
```

Required properties:

- compatible : Should be "fsl,s32g274a-pfeng"
- reg : Address and length of the register set for the device
- interrupts : Should contain all pfeng interrupts: hif0..hif3,nocpy,bmu, upegpt,safety
- clocks : Should contain at least: pfe_sys, pfe_pe
- memory-region : Physical address space for PFE buffers, must be in the range 0x00020000 - 0xbfffffff

Optional properties:

- fsl,fw-class-name : PFE CLASS firmware filename
- fsl,fw-util-name : PFE UTIL firmware filename
- fsl,pfeng-master-hif-channel : [slave only] The number of master's HIF channel (0-3)

Required subnode:

- ethernet : specifies the logical network interface

Requires properties for 'ethernet' subnode:

- compatible : Should be "fsl,pfeng-logif"
- reg : Small number, indexing the network interfaces
- phy-mode : See ethernet.txt file in the same directory
- fsl,pfeng-if-name : Logical interface name visible in the Linux
- fsl,pfeng-hif-channel : The number of used HIF channel (0-3)
- fsl,pfeng-eth-id : PFE EMAC id where the interface will be linked to

Optional properties for 'ethernet' subnode:

- clocks : List of clocks and associated names, each pointing at particular

- clock per interface typ. Clock must be named after interface type,
- with "tx_" as prefix, ie. "tx_rgmii". Note: RGMII delay suffixes must be omitted.
- pinctrl : List of handles and associated names, each pointing at a pin configuration node within a pin controller.
- fsl,pfeng-ihc : [master/slave only] Declares the interface for IHC traffic
 - Can be used only once within all ethernet subnodes
- local-mac-address : MAC address
- phy-handle : phandle to the PHY device connected to this device.
- fixed-link : Assume a fixed link. See fixed-link.txt in the same directory.
- Use instead of phy-handle.

Optional subnode for 'ethernet':

- mdio : specifies the mdio bus, used as a container for phy nodes according to phy.txt in the same directory

Requires properties for 'mdio' subnode:

- compatible = Should be "fsl,pfeng-mdio"

Example:

```
pfe@46080000 {
    compatible = "fsl,s32g274a-pfeng";
    reg = <0x0 0x46000000 0x0 0x1000000>, /* PFE controller */
        <0x0 0x4007ca00 0x0 0x4>, /* S32G274a syscon */
        <0x0 0x83400000 0x0 0xc00000>; /* PFE DDR 12M */
    #address-cells = <1>;
    #size-cells = <0>;
    memory-region = <&pfe_reserved>;
    interrupt-parent = <&gic>;
    interrupts = <0 190 1>, /* hif0 */
                <0 191 1>, /* hif1 */
                <0 192 1>, /* hif2 */
                <0 193 1>, /* hif3 */
                <0 194 1>, /* bmu */
                <0 195 1>, /* nocpy */
                <0 196 1>, /* upe/gpt */
                <0 197 1>; /* safety */
    interrupt-names = "hif0", "hif1", "hif2", "hif3",
                    "bmu", "nocpy", "upegpt", "safety";
    clocks = <&clks S32GEN1_CLK_PFE_SYS>,
            <&clks S32GEN1_CLK_PFE_PE>,
            <&clks S32GEN1_CLK_GMAC_0_TS>;
    clock-names = "pfe_sys", "pfe_pe", "pfe_ts";
    fsl,fw-class-name = "s32g_pfe_class.fw";
    fsl,fw-util-name = "s32g_pfe_util.fw";

    /* EMAC 0 */
    pfe0_if: ethernet@0 {
        compatible = "fsl,pfeng-logif";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0>; /* If id */
        local-mac-address = [ 00 04 9F BE EF 00 ];
    };
};
```

```

fsl,pfeng-if-name = "pfe0";
fsl,pfeng-hif-channel = <0>;          /* HIF channel 0 */
fsl,pfeng-ethernet-id = <0>;          /* EMAC 0 */
phy-mode = "sgmii";
phy-handle = <&anyphy1>;

/* MDIO on EMAC 0 */
pfe0_mdio: mdio@0 {
    /* on EVB occupied by USB ULPI */
    status = "disabled";
    compatible = "fsl,pfeng-mdio";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0x0>;
};

/* EMAC 1 */
pfel_if: ethernet@1 {
    compatible = "fsl,pfeng-logif";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <1>;                      /* If id */
    local-mac-address = [ 00 04 9F BE EF 01 ];
    fsl,pfeng-if-name = "pfel";
    fsl,pfeng-hif-channel = <1>;      /* HIF channel 1 */
    fsl,pfeng-ethernet-id = <1>;      /* EMAC 1 */
    phy-mode = "rgmii";
    phy-handle = <&anyphy2>;
    clocks = <&clks S32G274A_SCMI_CLK_PFE1_TX_SGMII>,
             <&clks S32G274A_SCMI_CLK_PFE1_TX_RGMII>,
             <&clks S32G274A_SCMI_CLK_PFE1_TX_RMII>,
             <&clks S32G274A_SCMI_CLK_PFE1_TX_MII>;
    clock-names = "tx_sgmii", "tx_rgmii",
                  "tx_rmii", "tx_mii";

    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl0_pfel_rgmii_c>,
                <&pinctrl11_pfel_rgmii_c>;

/* MDIO on EMAC 1 */
pfel_mdio: mdio@0 {
    compatible = "fsl,pfeng-mdio";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0x1>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl0_pfel_mdio_c>,
                <&pinctrl11_pfel_mdio_c>;
    anyphy1: ethernet-phy@1 {
        reg = <1>;
    };
    anyphy2: ethernet-phy@2 {
        reg = <2>;
    };
};

};

```

4.5 Performance consideration

To gain highest possible throughput on host interface, several options exist to enhance performance, loosing other features or resources:

- Double key ring size
 - Better throughput can be achieved by doubling HIF channel ring size. By default there is 256 entries in the ring. It's compile time option which must be changed in source code header file 'pfe_platform/public/pfe_platform_cfg.h', in named constant PFE_ - HIF_RING_CFG_LENGTH.
 - Restriction applies: not all interfaces can be enabled or reserved memory region has to be enlarged in device tree.
- Restrict RAM to 2GB (mem=2g)
 - Restrict Linux kernel memory to 2G limit. It minimizes bounce buffering on TX.
 - Disadvantage: waste memory.

4.6 Silicon cut 1.1 backward compatibility

By default, the driver is built for S32G2 cut 2.0. Older cut version contains some limitation which affects performance.

The driver is not able to autotune for different silicon versions, so users of cut 1.1 variant have to rebuild the driver manually.

The following two steps are necessary to build cut 1.1 compatible driver:


1. Adding compile time option 'PFE_CFG_IP_VERSION=PFE_CFG_IP_VERSION_NPU_7_14' (Note: without 'a' suffix as 7_14a means cut 2.0) enables all necessary errata support in driver.
2. Remove 'dma-coherent' property from driver's device tree node.

Chapter 5

Master-Slave feature

5.1 Prerequisite

The driver can run in multiple instances within the same system to allow sharing the PFE-provided connectivity using dedicated host interfaces. To enable Master-Slave, the driver gets extended to two binaries, *pfeng.ko* which additionally contains Inter-HIF-communication (aka 'IHC') support and which acts as Master and *pfeng-slave.ko*, which also contains IHC, but has lack of direct PFE controller manageability.

 *Master-Slave feature requires additional configuration in device tree*


5.2 Master

To build PFE driver with Master functionality, additional compile-time option is required:

```
make <options> PFE_CFG_MULTI_INSTANCE_SUPPORT=1 PFE_CFG_PFE_MASTER=1 ...
```

The Master requires marking one of the configured network interface for IHC transport in device tree by setting *fsl,pfeng-ihc* property, for example:

```
&pfe0_if {
    fsl,pfeng-ihc;
    fsl,pfeng-hif-channel = <0>;
    phy-mode = "sgmii";
    phy-handle = <&mdio_a_phy4>;
};
```

 *Only one network interface can carry 'fsl,pfeng-ihc' property*

5.3 Slave

To build PFE driver with Slave functionality, additional compile-time option is required:

```
make <options> PFE_CFG_MULTI_INSTANCE_SUPPORT=1 PFE_CFG_PFE_MASTER=0 ...
```

The Slave requires two configuration options in device tree:

- setting global configuration option 'fsl,pfeng-master-hif-channel' with Master's HIF channel number (see example in Master section). The option can be overwritten by command line parameter *master_ihc_chnl*.
- marking one of the configured network interface for IHC transport in device tree by setting *fsl,pfeng-ihc* property.

☞ *Only one network interface can carry 'fsl,pfeng-ihc' property'*

Slave example:

```
pfeslave@46000000 {
    compatible = "fsl,s32g274a-pfeng-slave";
    reg = <0x0 0x46000000 0x0 0x1000000>, /* PFE controller
        */
        <0x0 0x4007ca00 0x0 0x4>; /* S32G274a
        syscon */
    #address-cells = <1>;
    #size-cells = <0>;
    interrupt-parent = <&gic>;
    interrupts = <0 190 1>, /* hif0 */
        <0 191 1>, /* hif1 */
        <0 192 1>, /* hif2 */
        <0 193 1>, /* hif3 */
        <0 194 1>, /* bmu */
        <0 195 1>; /* nocpy */
    interrupt-names = "hif0", "hif1", "hif2", "hif3",
        "bmu", "nocpy";
    memory-region = <&pfe_reserved_slave>;
    fsl,pfeng-master-hif-channel = <0>;

    /* EMAC 1 */
    pfelsl_if: ethernet@1 {
        compatible = "fsl,pfeng-logif";
        #address-cells = <1>;
        #size-cells = <0>;
        status = "okay";
        reg = <1>;
        local-mac-address = [ 00 04 9F BE EF F1 ];
        fsl,pfeng-if-name = "pfelsl";
        fsl,pfeng-eth-id = <1>; /* Phy IF (EMAC) id */
        fsl,pfeng-ihc;
        fsl,pfeng-hif-channel = <3>;
        phy-mode = "internal";
    };
};
```

5.4 Running

When running a master-slave scenario, following points apply:

- Master driver must always be executed first to allow slave drivers to connect to it during their start-up phase.

- Slave driver(s) configure the PFE to ensure that packets can reach the driver. The PFE then distributes traffic to particular driver instances using destination MAC address of received packets following given rules:
 - Packets received via PFE EMAC interfaces with destination MAC address matching some running slave driver instance are delivered to that instance.
 - All remaining traffic is delivered to the master driver instance.
 - Default MAC address-based traffic distribution can be changed using FCI.
 - When master driver is reset, all slaves must be subsequently reset too

Chapter 6

IEEE1588 Support

The PFE is capable of providing precise, adjustable time reference with ability to timestamp ingress and egress PTP frames. Once enabled the driver ensures that ingress and egress PTP frames are timestamped at MAC level and provides timestamp to the stack.

Supported synchronization modes:

- End to End - SYNC, Follow_Up, Delay_Req, Delay_Resp
 - For UDP supported only on IP: 224.0.1.129, 224.0.1.130, 224.0.1.131, 224.0.1.132
- Peer to Peer - Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up
 - For UDP supported only on IP: 224.0.0.107

Transport layers:

- Ethernet - Supported
- UDP over IPv4 - Supportetd
- UDP over IPV6 - Currently not supported. Support will be added in future.

6.1 Clock configuration

S32G platform allows multiple clock configurations depicted in figure 6.1.

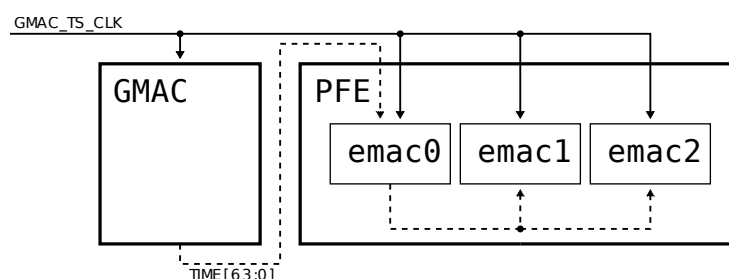


Figure 6.1: S32G2 IEEE1588 clock distribution

6.1.1 Internal Timestamp Mode(full line figure 6.1)

In this mode the PFE_EMAC is maintaining system time using internal timer clocked by reference signal and uses the timer value to timestamp frames or report current system time. GMAC_TS_CLK is used as reference. This is the only supported configuration. For this to work you have to add "pfe_ts" clock to the device tree.

Example (from *fsl-s32g274a.dtsi*):

```
clocks = <&clks S32G274A_SCMI_CLK_PFE_AXI>,
        <&clks S32G274A_SCMI_CLK_PFE_PE>,
        <&clks S32G274A_SCMI_CLK_PFE_TS>;
clock-names = "pfe_sys", "pfe_pe", "pfe_ts";
```

In case the "pfe_ts" is not provided the timestamping feature is disabled.

In Internal Timestamp Mode the GMAC_TS_CLK(RM clock name) resp. S32G274A_SCMI_CLK_PFE_TS(SCMI clock name) should be configured with respect to recommended frequency:

Mode	Minimum Frequency
10 Mbps Full Duplex	5 MHz
10 Mbps Half Duplex	5 MHz
100 Mbps Full Duplex	5 MHz
100 Mbps Half Duplex	5 MHz
1 Gbps Full Duplex	5 MHz
1 Gbps Half Duplex	18.75 MHz
2.5 Gbps Full Duplex	11.72 MHz
2.5 Gbps Half Duplex	46.875 MHz

6.1.2 External Timestamp Mode(dashed line figure 6.1)

The internal PFE_EMAC timer is disabled and PFE_EMAC is using time information delivered from external source. This source can be PFE_EMAC0 or the GMAC instance as shown in figure with the dashed line. NOTE: This configuration is currently not supported.

6.2 Driver interface

Driver implements standard Linux API for HW timestamping configuration.

6.2.1 IOCTL

The driver supports standard ioctl for timestamp configuration. This API is typically called by PTP stack.

- SIOCSHWTSTAMP - Set configuration (struct hwtstamp_config)
 - TX configuration: Timestamping on/off
 - RX configuration: Message specific configuration currently not supported, all P2P and E2E packets are always timestamped.
- SIOCGHWTSTAMP - Get configuration (struct hwtstamp_config)

6.2.2 PTP hardware clock device

After driver initialization there should be one device `/dev/ptpX` for each PFE_EMAC. Association of clock and interface is available in `ethtool` as "PTP Hardware Clock" number.

6.2.3 ethtool

Driver implements standard `ethtool` API for getting timestamp options via `ETHTOOL_GET_TS_INFO`.

Example output:

```
root@s32g274aevb:~# ethtool -T pfe0
Time stamping parameters for pfe0:
Capabilities:
    hardware-transmit      (SOF_TIMESTAMPING_TX_HARDWARE)
    software-transmit      (SOF_TIMESTAMPING_TX_SOFTWARE)
    hardware-receive       (SOF_TIMESTAMPING_RX_HARDWARE)
    software-receive       (SOF_TIMESTAMPING_RX_SOFTWARE)
    software-system-clock   (SOF_TIMESTAMPING_SOFTWARE)
    hardware-raw-clock     (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
    off                    (HWTSTAMP_TX_OFF)
    on                     (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
    none                   (HWTSTAMP_FILTER_NONE)
    some                   (HWTSTAMP_FILTER_SOME)
    ptpv1-l4-event         (HWTSTAMP_FILTER_PTP_V1_L4_EVENT)
    ptpv1-l4-sync          (HWTSTAMP_FILTER_PTP_V1_L4_SYNC)
    ptpv1-l4-delay-req     (HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ)
    ptpv2-l4-event         (HWTSTAMP_FILTER_PTP_V2_L4_EVENT)
    ptpv2-l4-sync          (HWTSTAMP_FILTER_PTP_V2_L4_SYNC)
    ptpv2-l4-delay-req     (HWTSTAMP_FILTER_PTP_V2_L4_DELAY_REQ)
    ptpv2-l2-event         (HWTSTAMP_FILTER_PTP_V2_L2_EVENT)
    ptpv2-l2-sync          (HWTSTAMP_FILTER_PTP_V2_L2_SYNC)
    ptpv2-l2-delay-req     (HWTSTAMP_FILTER_PTP_V2_L2_DELAY_REQ)
    ptpv2-event            (HWTSTAMP_FILTER_PTP_V2_EVENT)
    ptpv2-sync             (HWTSTAMP_FILTER_PTP_V2_SYNC)
    ptpv2-delay-req        (HWTSTAMP_FILTER_PTP_V2_DELAY_REQ)
```

6.3 BSP yocto support

Hint: add package "linuxptp" to your Yocto configuration file. With this configuration you get access to Linux PTP stack "ptp4l" and clock synchronization demon "phc2sys" for advanced configurations.

```
IMAGE_INSTALL_append = " linuxptp"
```

Chapter 7

LibFCI

7.1 The libfci

Additional features provided by the PFE can be managed using FCI API. The driver release package contains library sources which can be used to control PFE from custom user's application. Details about usage of the library can be found in '*FCI API Reference*' document.

7.2 Building libfci library

The simplest way, how to get in touch with libfci framework, is using the Auto Linux BSP facilities. The BSP contains libfci recipe in `meta-alb/recipes-extended` subdirectory, which is intended for quick jump-in to the FCI development. By default, the libfci is not enabled, so the first step is to create libfci.a library:

```
user@dev:~/fsl-auto-yocto-bsp/build_s32g274aevb$ bitbake -c compile libfci
```

7.3 Entering Yocto development environment

Yocto offers extra task called devshell. The task will deposit all the libfci source code into a directory, apply all patches included in the recipe, and then open a terminal in that directory:

```
user@dev:~/fsl-auto-yocto-bsp/build_s32g274aevb$ bitbake -c devshell libfci
```

When invoked, all environmental variables are set up exactly like for compilation. Use predefined `$CC`, `$CXX`, `$PATH`. The libfci.a library, which was compiled in previous step, is located in `sw/xfci/libfci/build/release` directory, what can be checked by `find` command, inside devshell terminal:

```
root@dev:~/fsl-auto-.../libfci/0.9.3-r0/git# find . -name libfci.a
./sw/xfci/libfci/build/release/libfci.a
```

The same way it can be checked for required header files:

```
root@dev:~/fsl-auto-.../libfci/0.9.3-r0/git# find . -name libfci.h
./sw/xfci/libfci/public/libfci.h
```

The above information is enough to have all necessary bits for libfci-enabled application development.