WiFi Driver for Freescale MQX

Reference: GS_WiFiDriver

Version: SP-1.2

Date: *18-May-11*

Version	Date	Remarks
1.0	14 Dec 2010	Initial release.
1.1	05 April 2011	K60N512 Tower Module support added
1.2	17 May 2011	IAR IDE support added

GainSpan Corporation
125 South Market Street, Suite 400
San Jose, CA 95113
U.S.A.

+1 (408) 673-2900

info@GainSpan.com
www.GainSpan.com

Table of Contents

1 Overview

1.1 Purpose

This document describes the driver source code and the related APIs for the Freescale MQX RTOS, to integrate it with Gainspan GS1011 Wi-Fi module.

.

1.2 Scope

Scope of this document is to explain the GainSpan Wi-Fi Driver APIs in MQX operating system which runs on the Freescale ColdFire (MCF52259) and K60 (MK60N512VMD100) processor. This document does not explain the design of the pre-existing MQX components that may be present in the system.

1.3 Overview

The IP2WIFI stack on the GainSpan GS1011M is used to provide Wi-Fi Capability to any devices having serial interface. The serial interface can be either UART or SPI. These serial hosts use simple command interface to configure the IP2WIFI Adapter (GainSpan GS1011M) and to create the wireless connections.

1.4 Terminology

Term	***Explanation***
API	Application Programmer's Interface
GS	GainSpan
CS	Chip Select
UART	Universal Asynchronous Receiver/Transmitter
SPI	Serial Peripheral Interface
CONT	Continuous
Host	Refers Microcontroller(Freescale processor) with master SPI
Node	Refers to Freescale tower board with GS1011M
Octet	8 bit data
RX	Receive
TX	Transmit
XOFF	Transmit OFF
AP	Wireless Access Point.
SOC	System On Chip
SW	Software
HW	Hardware
XON	Transmit ON

1.5 Reference

- GS IP2WIFI adaptor guide.
- TWR-WIFI-G1011M User Manual
- TWR-K60N512 Tower Module User Manual Rev. 1.0
- Freescale Coldfire documents (www.freescale.com/mcf5225x)
- K60 Sub-Family Reference Manual / Document Number: K60P144M100SF2RM (*www.freescale.com/Kinetis)*
- Freescale MQX 3.7 reference documents (www.freescale.com/mqx)

2 Design Considerations

2.1 Usage Scenario

This Wi-Fi driver connects the Freescale MQX 3.6/ MQX 3.7 network stack to the wireless interface provided by the GS1011M.

2.2 Objective

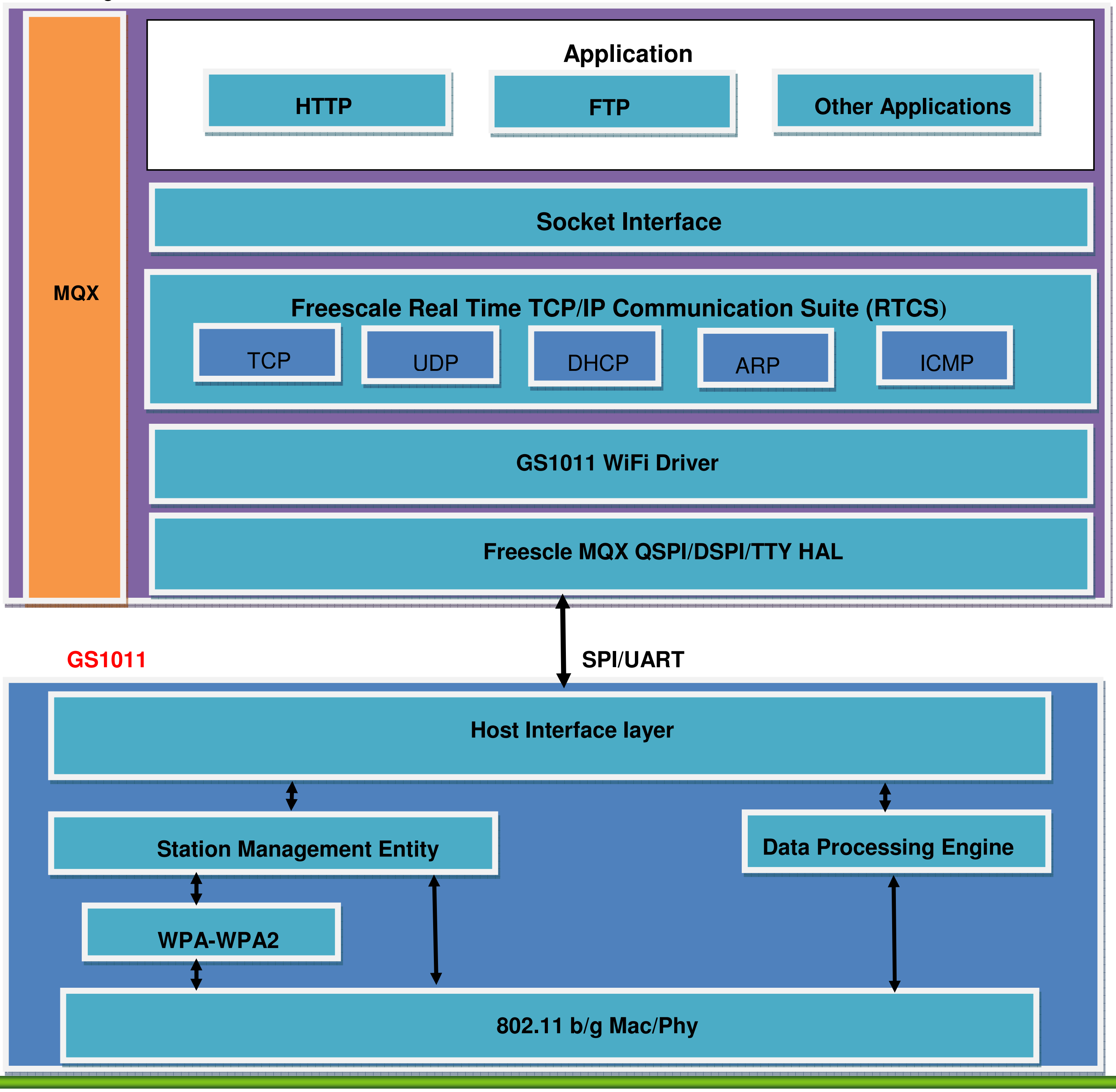
The primary objectives of this design document are to address implementation of the Wi-Fi driver on the Freescale MQX 3.6/MQX 3.7 operating system and its interface to the IP2WIFI firmware running on the GainSpan GS1011M module.

2.3 Dependencies

- The interface of IP2WIFI stack running on the GS1011M supports both UART and SPI. Use of UART or SPI depends on the state of the GPIO26 pin of the GS1011M. If GPIO26 is pulled HIGH, and then it is SPI, if it is a no-connect or pulled LOW, it is UART. Default interface is SPI for the TWR-WIFI-G1011MI and TWR-K60N512 Tower Module board. Refer the TWR-WIFI-G1011M and TWR-K60N512 Tower Module user Manual for jumper setting.
- The default interface is SPI and compile time option shall be provided to select UART interface. The compile time flag is GS_SPI_PORT_ENABLE and it is defined in gs_prv.h. If this #define flag is used then SPI is enabled, else if this flag is commented out, then UART is used to communicate with GS1011M board.
- The UART Baud rate is 115200 with no HW/SW flow support.
- With SPI interface by default SPI Mode#0 is selected (CPOL =0 and CPH=0)
- With SPI interface one GPIO is reserved for GS1011M Slave data ready detection [B62 (IRQ_A)]
- SPI supports only Motorola mode with 8-bit SPI data word size
- Max SPI clock rate supported in burst mode is 1200KHz
- With SPI interface the GS1011M SPI always runs as a slave device

3 Software Architecture Overview

The following figure illustrates the software architecture of the Gainspan wifi module(GS1011) integrated with a Freescale microcontroller.



As shown in the figure above, the GS1011 module is integrated with the Freescale Host Microcontroller using SPI or UART.
The Host transmits/receives raw data using SPI or UART interface when the GS1011 module is configured for SPI/UART mode. A thin driver on the Host takes care of interacting with the Wi-Fi module through the SPI or UART Host interface.

The following sections explain in brief the various components illustrated in the above figure.

3.1 Host

The host is any system that has applications being executed and has an SPI or UART interface to connect to the GS1011 Wi-Fi module.

3.1.1 SPI

The SPI on the Host side provides an interface for the Host to access the Wi-Fi module.
SPI on the Host acts as the Master.

3.1.2 UART

The UART on the host side interface for the Host to access the Wi-Fi module, the baud rate of the UART interface is 115200

3.2 GS1011 Module

The GS1011 module incorporates Wi-Fi functionality.

3.2.1 SPI

The SPI on the GS1011 module acts as a SPI slave. It is a standard 4-wire SPI and can support a maximum frequency of 3 MHz.

3.2.2 UART

The UART on the GS1011 module is a standard UART interface with a baud rate of 115200

3.2.3 Host Interface Layer

This layer abstracts the lower layers in the host interface with which the GS1011 module is connected. This layer interacts with the station management layer for wireless control actions and data processing layer for IP, Arp and DHCP packet transmission. This is the AT command layer of GS1011 module.

3.2.4 Station Management Entity

This is the core layer which manages the Wi-Fi connectivity. This maintains the state machine to detect the activity on the Wi-Fi network and indicates to the user accordingly. It interacts with the WPA supplicant if Security is enabled in the Wi-Fi network.

3.2.5 WPA-WPA2

The WPA supplicant is used to initiate the 802.1x/EAP authentication if WPA/WPA2-PSK is used as the security parameter. It also plays a major part in performing the 4-way handshake to derive the PTK in WPA/WPA2-PSK modes.

3.2.6 Data Processing Engine

The layer processes data obtained from the Host or from the network. The functioning of this layer depends on the direction and type of the frame. This layer interacts with the host through the host interface layer which uses ESC sequence to send –receive data from host.

4 GS1011 Driver Interfaces

4.1 Driver Design Descriptions

This section describes the GS1011 Wi-Fi driver implementation in details..

4.2 Initialization

The initialization of the GS Wi-Fi driver starts with the Gs_initialize() function which is getting called from the Freescale network stack. This function initializes the serial interface on the Freescale processor, allocating all resources and sends some commands to get the basic information of the GS SOC.

This initialization function creates the necessary task to handle the serial data/response from the GS SOC also. Once initialization is done this function puts the node into the ready state so the Wi-Fi driver can accept any other APIs.

The interface structure between the GS WIFI driver and Freescale MQX network stack is Gs_IF of type ENET_MAC_IF_STRUCT. This structure contains the entire WIFI driver APIs that the network stack can access each one through this structure.

4.3 Serial input handling

For handling the serial data from GS1011 module, a separate task has been created during initialization. This task receives all the serial incoming data from the GainSpan soc and process.

Task function: gs_serial_task(uint_32 initial_data).

The serial incoming data are three types.

- AT Command response.
- Data.
- Asynchronous message.

This task runs as a state machine. This task process each command response, signaling the task which is waits for the response and store or forwards that information to the application.

The data comes from the GS SOC is packed with a header and tail. This serial input task strip the header and pass the data to the MQX network stack.

The data coming from GS SOC in the following format:

ESC<R>:<Length>:<Data>

4.4 SPI Data Handler Task:

In case of SPI interface additional task is required to handle SPI data transfer. SPI data transfer always works in full duplex mode.

Task function: gs_spi_hlr_task ()

SPI data transfer request will be triggered by following events:

- **Data transfer request by Host Application Task (write request):** After receiving the data transfer request from application task via write operation, SPI data handler task starts transmitting data to slave GS1011 node, till all the data transferred and host wake-up signal is in de-asserted state(host wake-up signal becomes LOW). While transmitting the data, it also receives data from S2w Application, resulting in full-duplex transfer.
- **Data ready indication from Slave GS101 node via host wake-up signal:** Host wake-up signal assertion (LOW to HIGH transition) is detected by GPIO driver and appropriate callback will be executed to send an event to SPI handler task. After receiving host wake-up signal event, SPI handler task starts pulling out the data by giving clock (i.e. write IDLE dummy pattern). It continues to pull the data as long as Host wake-up signal is HIGH. During this period, if any transmit request is pending, that's getting serviced. This approach results in full duplex data transfer.

During data transfer it also handles flow control .If there is no data to transmit then IDLE fill pattern shall be transmitted.

Byte stuffing and de-stuffing is incorporated by this task. Refer Appendix section for more details on SPI data transfer.

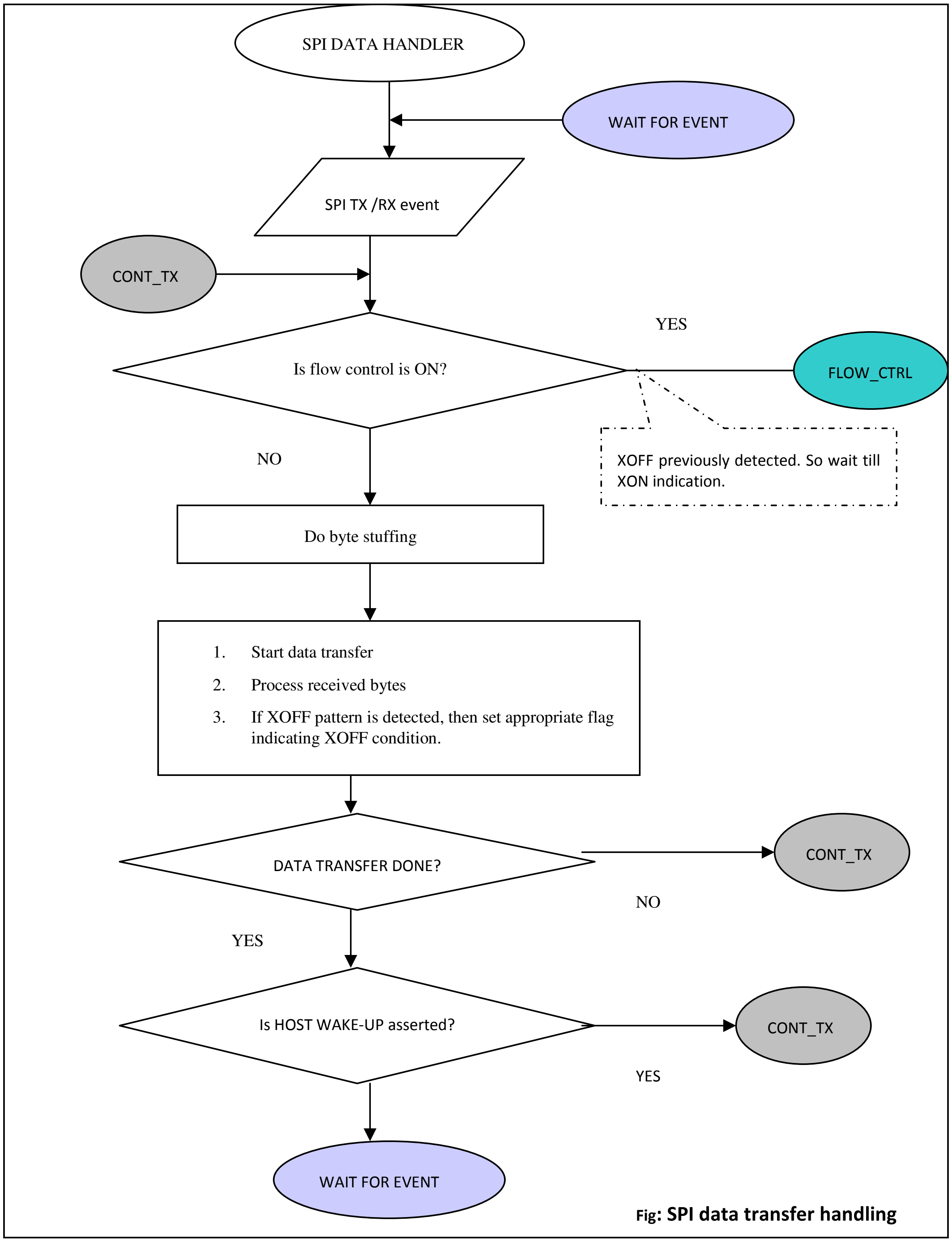

SPI DATA HANDLER
WAIT FOR EVENT
SPI TX /RX event
CONT_TX
Is flow control is ON?
YES
FLOW_CTRL
XOFF previously detected. So wait till XON indication.
NO
Do byte stuffing
1. Start data transfer
2. Process received bytes
3. If XOFF pattern is detected, then set appropriate flag indicating XOFF condition.
DATA TRANSFER DONE?
CONT_TX
NO
YES
Is HOST WAKE-UP asserted?
CONT_TX
YES
WAIT FOR EVENT
Fig: SPI data transfer handling

Figure: SPI data handling flow chart.

4.5 Data Send handling

The Wi-Fi driver provides glue logic to interface the data send from MQX network stack to the IP2WIFI stack running on the GS SOC. This glue logic adds the header to the data comes from the MQX network stack and sends to the GS SOC.

The data send format to the GS SOC is:

ESC<R>:<Length>:<Data>

4.6 Connection management handling

The connection management state machine starts with the Wi-Fi mode setting on the Wi-Fi driver using the GS Wi-Fi driver ioctl (gs_mediactl()) function. Before setting the mode the application should fill the ssid, security type and security key/information on the GS Wi-Fi driver using the GS Wi-Fi driver ioctl(gs_mediactl()) functions.

The connection management state machine function is gs_conn_handler(). The main states of this connection management are:

- Station Ready
- Station Down, not associated
- Station security set done
- Station management done
- Station association done.

This connection management handling any disconnect event from the AP and make sure the node is being associated to the AP.

4.7 Command response handling

The commands to the GS SOC are simple AT commands to access the GS IP2WIFI stack running on the GS SOC. These commands are issued by the application which internally calls the GS WIFI driver APIs to communicate to the GS SOC.

Commands are basically of two types:

1. Set command.

   This type basically configures the GS SOC with the application specific parameters
2. Get Command.

   This type is used to get the configured information from the GS SOC.

All commands are mapped by a corresponding driver APIs and can access using the driver ioctl (gs_mediactl) function.

5 Driver Callbacks for ENET Layer

This section describes the GS1011 Driver Callbacks for the ENET Layer

5.1 Gs_initialize

5.1.1 Description

During the initialization, GS1011 driver associates itself with the ENET layer, allocates private area for the driver control block, prepares the PCB pool for host-receive packets, creates the driver task for handling the events from GS1011 module which also include receiving packets, registers the interrupt handler (IRQ Falling) to receive the interrupt requests from GS1011 module, setting up the GS1011 module (firmware boot loading, module SPI initialization, etc).

5.1.2 Prototype

uint_32 Gs_initialize (ENET_CONTEXT_STRUCT_PTR enet_ptr);

5.1.3 Paramters

enet_ptr : The Ethernet state structure.

5.1.4 Return type

The status of operation, either ENET_OK (on success) or ENET_ERROR (on failure) is returned.

5.2 Gs_stop

5.2.1 Description

This API reset the GS1011 and free all the resources allocated for the GS1011 Wi-Fi driver

5.2.2 Prototype

uint_32 Gs_stop (ENET_CONTEXT_STRUCT_PTR enet_ptr);

5.2.3 Paramters

- enet_ptr : The Ethernet state structure.

5.2.4 Return type

The status of operation, either ENET_OK (on success) or ENET_ERROR (on failure) is returned.

5.3 Gs_send

5.3.1 Description

The Gs_send handler is responsible for packet transmission. The packet to be transmitted is sent to the GS1011 module only if the wireless link setup is finished and the GS1011 module is not running out of buffers. In either of the cases error is returned. The packet to be transmitted is sent to the GS1011 module through SPI or UART. This is done in the calling task (TCPIP task) context itself. If the packet is successfully sent to the GS1011 module, the Packet Control Block (PCB) and corresponding packet memory is freed up, otherwise error is returned.

5.3.2 Prototype

```
uint_32 Gs_send (
    ENET_CONTEXT_STRUCT_PTR enet_ptr,
    PCB_PTR packet,
    uint_32 size,
    uint_32 frags,
    uint_32 flags
);
```

5.3.3 Parameters

- enet_ptr : The Ethernet state structure.
- packet: the packet to send
- size : total size of the packet
- frags: total fragments in the packet
- flags: optional flags, zero = default

5.3.4 Return type

The return type for this handler will be one of the following:

- ENET_OK on success

- ENETERR_INVALID_DEVICE is returned when driver control block could not be retrieved
- ENETERR_SEND_FULL is returned when the packet cannot be transmitted
- ENETERR_SEND_SHORT is returned when the packet is smaller than Ethernet header length
- ENETERR_SEND_LONG is returned when the packet is bigger than ENET_FRAME_SIZE

5.4 Gs_mediactl

5.4.1 Description

The Gs_mediactl handler of the GS1011 driver provides the interface to configure and control wireless connection parameters.

5.4.2 Prototype

```
uint_32 Gs_ mediactl (
  ENET_CONTEXT_STRUCT_PTR enet_ptr,
  uint_32 command_id,
  pointer inout_param
);
```

5.4.3 Parameters

- enet_ptr : The Ethernet state structure.
- command_id : The particular control operation identifier
- inout_param : Data for the command

The supported command_id are:

- ENET_SET_MEDIACTL_MODE

  This command saves the mode of the Wi-Fi as either infrastructure or *ad hoc* and starts the Wi-Fi association.
- ENET_SET_MEDIACTL_SCAN

  This command starts the scan of all Wi-Fi networks present in the vicinity.
- ENET_SET_MEDIACTL_PASSPHRASE

  This command set the wpa/wpa2 passphrase of the Wi-Fi module.
- ENET_SET_MEDIACTL_ESSID

This command sets the ssid to which the Wi-Fi module to be associated.

- ENET_SET_MEDIACTL_RETRY

  This command set the retry count of the Wi-Fi module.

- ENET_SET_MEDIACTL_ENCODE

  This command set the wep authentication details of the Wi-Fi module.

- ENET_SET_MEDIACTL_POWER

  This command enable/disable the power save mode of the Wi-Fi module.

- ENET_SET_MEDIACTL_SEC_TYPE

  This command sets the security type of the Wi-Fi module.

- ENET_SET_WPS_ENABLE

  This command starts the wps push button method of the Wi-Fi module.

- ENET_SET_WEB_PROV_ENABLE

  This command starts the web provisioning feature of the Wi-Fi module.

- ENET_SET_WEB_PROV_PARAM_SSID

  This command set the ssid of the ad hoc network to be created for the web provisioning feature of the Wi-Fi module.

- ENET_SET_WEB_PROV_PARAM_CH

  This command set the channel of the ad hoc network to be created for the web provisioning feature of the Wi-Fi module.

- ENET_SET_WEB_PROV_PARAM_USRNAME

  This command set the username authentication of webserver to be created on the Wi-Fi module for the web provisioning feature.

- ENET_SET_WEB_PROV_PARAM_PWD

  This command set the password authentication of webserver to be created on the Wi-Fi module for the web provisioning feature.

- ENET_GET_MEDIACTL_SEC_TYPE

  This command return the security type of the Wi-Fi module configured.

- ENET_SET_MEDIACTL_PASSPHRASE

  This command returns the wpa/wpa2 passphrase of the Wi-Fi module configured.

- ENET_GET_MEDIACTL_MODE

  This command returns the mode of the Wi-Fi module configured.

- ENET_GET_MEDIACTL_SCAN

This command returns the scan details of all Wi-Fi networks present in the vicinity.

- ENET_GET_MEDIACTL_RETRY

  This command returns the retry count of the Wi-Fi module configured.

- ENET_GET_MEDIACTL_ENCODE.

  This command returns the WEP authentication details of the Wi-Fi module configured.

- ENET_GET_MEDIACTL_POWER

  This command returns the power save mode of the Wi-Fi module configured.

- ENET_MEDIACTL_IS_INITIALIZED

  This command returns the state of the Wi-Fi module initialized as either true or false.

- ENET_GET_WEB_PROV_PARAM_USRNAME

  This command returns the username authentication of web server to be created on the Wi-Fi module for the web provisioning feature.

- ENET_GET_WEB_PROV_PARAM_PWD

  This command returns the password authentication of web server to be created on the Wi-Fi module for the web provisioning feature.

5.4.4 Return type

The status of operation, either ENET_OK (on success) or ENET_ERROR (on failure) is returned.

5.5 phy_gs_get_link_status

5.5.1 Description

This callback reports the link status when queried.

5.5.2 Prototype

uint_32 phy_gs_get_link_status (ENET_CONTEXT_STRUCT_PTR enet_ptr);

5.5.3 Parameters

enet_ptr : The Ethernet state structure.

5.5.4 Return type

Returns zero if link is up, else a non zero value

The following APIs are dummy functions.

5.6 Gs_read

5.7 Gs_write

5.8 Gs_join

5.9 Gs_rejoin

6 Media Control Interfaces

This section describes the Media Control Interfaces and how the GS1011 module can be configured using these interfaces.

The Free scale RTCS provides the Wireless media parameters control and configuration interfaces through the iwconfig set/get API or *ENET_mediactl* API.

All the control and configuration commands issued by the application for a particular device are directed to the media control callback submitted by that device driver. For the GS1011 module, *Gs_mediactl* is the callback.

The GS1011 device driver's set/get functions are invoked by the *Gs_mediactl* in response to the iwconfig/ENET_mediactl invocations from the application.

The following are the parameters that can be configured:

- Power save mode
- Wireless retry count(both set and get)
- Scan (both set and get)
- Security type (both set and get)
- Network to join (essid) (both set and get)
- Network mode(both set and get)
- Wpa/wpa2 psk security passphrase (both set and get)
- Wep key(both set and get)
- Wps
- Web provisioning

Arguments for some of the above parameters are defined in enet_wifi.h.

The iwconfig APIs for setting the power save model, getting the scan results and setting transmit rate are provided as part of the RTCS. These APIs have been added in the iwcfg.c file (rtcs\source\if). In order to make these APIs available for the application, the RTCS has to be re-built by including these additions.

7 Network Interface

The GS1011 device driver interfaces with the ENET layer of RTCS as an NIC. The GS1011 module is added as the ENET device 1, whereas the Ethernet controller (FEC) is ENET device 0. The MAC and PHY callbacks and the default ENET device parameters submitted for GS1011 module are as follows:

- Gs_IF global structure is declared in mqx/source/io/enet/gs/gs_init.c file.

```
const ENET_MAC_IF_STRUCT Gs_IF = {
            Gs_initialize,
            Gs_stop,
            Gs_send,
            Gs_read,
            Gs_write,
          #if BSPCFG_ENABLE_ENET_MULTICAST
            Gs_join,
            Gs_rejoin,
          #endif
            Gs_mediactl
};
```

- phy_gs_IF global structure is declared in mqx/source/ source/io/enet/phy/phy_gs.c file.

```
const ENET_PHY_IF_STRUCT phy_gs_IF = {
        phy_gs_discover_addr,
        phy_gs_init,
        phy_gs_get_speed,
        phy_gs_get_link_status
};
```

8 GS1011 Driver integration with BSP

8.1 Integration Instructions

Following sections describes steps needed for porting GS WiFi driver to MQX BSPs.

8.2 TWR-WIFI-G1011MI tower module with MCF52259 processor:

The GS1011 driver is integrated with MQX version 3.7. The CodeWarrior version is “CW for Microcontrollers v 10.1”. All the directory paths mentioned are with respect to C:\Program Files\Freescale\Freescale MQX 3.7. The following are the steps followed to enable the GS1011 module support in the BSP.

In the enet_ini.c (path mqx/source/bsp/twrmcf52259/enet_ini.c), declare the following variables:

```
#if BSPCFG_ENABLE_GS
const ENET_IF_STRUCT ENET_1 = {
        &Gs_IF,
        &phy_gs_IF,
        1,
        1,
        0,
};
GS_PARAM_WIFI_STRUCT gs_wifi_param = {
  PARAM_CONFIG_ESSID,
  PARAM_CONFIG_PASSPHRASE,
  PARAM_CONFIG_PASSPHRASE_LEN,
  PARAM_CONFIG_ESSID_LEN,
  PARAM_CONFIG_SECURITY,
  PARAM_CONFIG_REG_DOMAIN,
  PARAM_CONFIG_CHANNEL,
  PARAM_CONFIG_PS_MODE,
  PARAM_CONFIG_IW_MODE,
  BSP_GS_SPI_DEVICE,
   BSP_GS_GPIO_DEVICE,
   BSP_GS_GPIO_INT_PIN
};
#endif
```

Next, edit the ENET_default_params as follows to add an entry for the GS1011 module

```
const ENET_PARAM_STRUCT ENET_default_params[BSP_ENET_DEVICE_COUNT] = {
 {
    &ENET_0,
    Auto_Negotiate,
    0,
    BSPCFG_TX_RING_LEN,   // # tx ring entries
    BSPCFG_TX_RING_LEN,   // # large tx packets
    ENET_FRAMESIZE,         // tx packet size
    BSPCFG_RX_RING_LEN,   // # rx ring entries
    BSPCFG_RX_RING_LEN,   // # normal rx packets - must be >= rx ring entries
    ENET_FRAMESIZE,      // ENET_FRAMESIZE,   // rx packet size
    BSPCFG_RX_RING_LEN,   // # rx PCBs - should be >= large rx packets.
    0,
    0,
    NULL
 }
 #if BSPCFG_ENABLE_GS
 ,
 {
    &ENET_1,
    // # Default WiFi Device parameter
    Auto_Negotiate,
    0,
    BSPCFG_TX_RING_LEN,   //  # NOT USED IN GS1011
    BSPCFG_TX_RING_LEN,   //  # NOT USED IN GS1011
    ENET_FRAMESIZE,      //  # NOT USED IN GS1011
    BSPCFG_RX_RING_LEN,   //  # NOT USED IN GS1011
    BSPCFG_RX_RING_LEN,   //  # NOT USED IN GS1011
    ENET_FRAMESIZE,      //  # NOT USED IN GS1011
    BSPCFG_GS_PCB,    //  # rx PCBs
    0,
    0 ,
    (pointer)&gs_wifi_param
 }
 #endif
};
```

Include the gs_prv.h and phy_gs.h headers in this file. Place the driver files in the directory mqx/source /io/enet/ and add them to the BSP project.

Add / Modify the following defines in the mqx/source/bsp/twrmcf52259/twrmcf52259.h file

```
#define BSP_ENET_DEVICE_COUNT  (MCF5XXX_FEC_DEVICE_COUNT+
                          (BSPCFG_ENABLE_GS?1:0))
/* GainSpan wifi Device configuration macro. */
#ifndef BSPCFG_ENABLE_GS
   #define BSPCFG_ENABLE_GS        0
#endif
#ifndef BSPCFG_GS_PCB
   #define BSPCFG_GS_PCB          16
#endif
```

Define the BSPCFG_ENABLE_GS macro in the config\twrmcf52259\user_config.h file as follows:

```
#define BSPCFG_ENABLE_GS 1
```

Enable the SPI0 device and GPIO inclusion into the BSP (this is disabled by default), as below in the config\twrmcf52259\user_config.h file

```
#define BSPCFG_ENABLE_SPI0        1
#define BSPCFG_ENABLE_GPIODEV    1
```

Define the following macros in mqx/source/bsp/twrmcf52259/twrmcf52259.h file

```
#define BSP_GS_SPI_DEVICE                "spi0:"
#define BSP_GS_GPIO_DEVICE                "gpio: input"
#define BSP_GS_GPIO_INT_PIN              (GPIO_PORT_NQ|GPIO_PIN_IRQ_RISING|GPIO_PIN1)
```

Now build the BSP followed by the RTCS and the application

8.3 TWR-K60N512 tower module with MK60N512VMD100 processor:

The GS1011 driver is integrated with MQX version 3.7.

Tower Module	IDE
TWR-K60N512	**IAR** Embedded Workbench version **6.20**

The following files must be modified to enable the GS1011 module support in the K60N512 BSP.

S. No.	File to modify
1.	mqx/source/bsp/twrk60n512/init_enet.c
2.	mqx/source/bsp/twrk60n512/twrk60n512.h
3.	config/twrk60n512/user_config.h
4.	

8.3.1 ENET parameter update

➢ **File Name: init_enet.c**

a) **Include GS driver specific header files:**

```
#if BSPCFG_ENABLE_GS
    #include <gs_prv.h>
    #include "phy_gs.h"
 #endif
```

b) **Update ENET_default_params[] structure :**

```
#if BSPCFG_ENABLE_GS
 const ENET_IF_STRUCT ENET_1 = {
    &Gs_IF,
    &phy_gs_IF,
    1,
    1,
    0,
};

GS_PARAM_WIFI_STRUCT gs_wifi_param = {
  PARAM_CONFIG_ESSID,
  PARAM_CONFIG_PASSPHRASE,
  PARAM_CONFIG_PASSPHRASE_LEN,
  PARAM_CONFIG_ESSID_LEN,
  PARAM_CONFIG_SECURITY,
  PARAM_CONFIG_REG_DOMAIN,
  PARAM_CONFIG_CHANNEL,
  PARAM_CONFIG_PS_MODE,
  PARAM_CONFIG_IW_MODE,
  BSP_GS_SPI_DEVICE,
   BSP_GS_GPIO_DEVICE,
   BSP_GS_GPIO_INT_PIN
};
#endif

const ENET_PARAM_STRUCT ENET_default_params[BSP_ENET_DEVICE_COUNT] = {
   {
      &ENET_0,
      Auto_Negotiate,
      ENET_OPTION_RMII,
      BSPCFG_TX_RING_LEN,   // # tx ring entries
      BSPCFG_TX_RING_LEN,   // # large tx packets
      ENET_FRAMESIZE,       // tx packet size
      BSPCFG_RX_RING_LEN,   // # rx ring entries
      BSPCFG_RX_RING_LEN,   // # normal rx packets - must be >= rx ring entries
      ENET_FRAMESIZE,       // ENET_FRAMESIZE,   // rx packet size
      BSPCFG_RX_RING_LEN,   // # rx PCBs - should be >= large rx packets.
      0,
      0
   }
#if BSPCFG_ENABLE_GS
      ,
       {
        &ENET_1,
        Auto_Negotiate,
        0,
        BSPCFG_TX_RING_LEN,
        BSPCFG_TX_RING_LEN,
        ENET_FRAMESIZE,
        BSPCFG_RX_RING_LEN,
        BSPCFG_RX_RING_LEN,
        ENET_FRAMESIZE,
        BSPCFG_GS_PCB,              //  # rx PCBs
        0,
        0 ,
        (pointer)&gs_wifi_param
      }
#endif
};
```

8.3.2 GS WI-FI Driver specific configuration update

- **File Name: twrk60n512.h**

a) Newly added Macro list:

```
#ifndef BSPCFG_ENABLE_GS
    #define BSPCFG_ENABLE_GS           0          /* If GS driver module is NOT enabled */
#endif

#if BSPCFG_ENABLE_GS
    #define BSP_GS_UART_DEVICE         "ittye:"         /* GainSpan UART channel configuration */
    #define BSP_GS_SPI_DEVICE          "spi2:"          /* GainSpan SPI channel */
    #define BSP_GS_GPIO_DEVICE         "gpio:input"     /* GPIO pin as input*/
   /* GPIO pins configuration PORT A pin#27 */
   #define BSP_GS_GPIO_INT_PIN         (GPIO_PORT_A|GPIO_PIN_IRQ_RISING|GPIO_PIN27)
   #ifndef BSPCFG_GS_PCB
        #define BSPCFG_GS_PCB          16               /* PCB count for GS driver module */
    #endif
#endif
```

b) Existing Macro's modified:

```
  /* ENET device count added for GS driver */
 #define BSP_ENET_DEVICE_COUNT       (MACNET_DEVICE_COUNT+ (BSPCFG_ENABLE_GS?1:0))
```

- **File Name**: user_config.h

a) Newly added Macro list:

```
 #define BSPCFG_ENABLE_GS        1    /* New macro defined to enable GS driver */
```

b) Modified Macro list:

```
#define BSPCFG_ENABLE_ITTYE      1   /* serial interface to GS1011 SOC */
#define BSPCFG_ENABLE_TTYF       0   /* Not used in case of GS1011 SOC interface */
#define BSPCFG_ENABLE_SPI2       1   /* SPI interface to GS1011 SOC */
#define BSPCFG_ENABLE_FLASHX     1
```

Now build the BSP project followed by the RTCS and the application.

8.4 GS1011 and MCU Interface

The default interface between the MCU and GS1011 module is SPI. To enable the UART interface modify the following files.

Processor	File Name	Modifications	Remarks
MCF52259	config\twrmcf52259\user_config.h	#define BSPCFG_ENABLE_ITTYC 1	
K60	config\twrk60n512\user_config.h	#define BSPCFG_ENABLE_ITTYE 1	
ANY	mqx\source\io\enet\gs\gs_prv.h	//#define GS_SPI_PORT_ENABLE	Comment out the macro *GS_SPI_PORT_ENABLE*

8.5 GS1011 Driver Source Code

To integrate the GS1011 device driver with the BSP, GS1011 driver source code has been placed in the enet directory [mqx\source\io\enet] along with the other network device drivers under the directory name “gs”. This source code contains the MAC interfaces and definitions of GS1011 device driver to the ENET layer. The PHY related source and header for the GS1011 module have been placed in the phy directory [mqx\source\io\enet\phy]. These files contain the phy interfaces to the ENET layer.

The other BSP interface files have been placed under the below mentioned directory:

Target processor	BSP files directory	Remarks
MCF52259	mqx\source\bsp\ twrmcf52259	
K60N512	mqx\source\bsp\twrk60n512	

All the above mentioned files and directories are similarly added to the bsp_twrmcf52259/ bsp_twrk60n512 project [mqx\build].

phy_gs.c and phy_gs.h are the files that go into phy directory [mqx\source\io\enet\phy]. These files must be added to bsp_twrmcf52259/bsp_twrk60n512 project.

In the gs directory,

Gs_init.c contains the driver initialization, interface initialization and resource allocation for the GS1011 wifi driver.

Gs_open.c contains the driver open function

Gs_utils.c contains various functions which communicate to the GS1011 module through SPI or uart interface with predefined command set.

Gs_send.c contains the data send function which sends data from MCU to GS1011 module.

Gs_receive.c contains the data receive function which receive data from GS1011 module pass to the MCU network stack.

Gs_conn_mgt.c contains the connection management functions.
Gs_con_manage.c contains the network connection handler.

gs_mediactl.c contains the media control handlers

Gs_prv.h contains the GS1011 module specific definitions.

9 Appendix A: Example Usage in the Application

To enable the GS1011 NIC, the application needs to invoke the following APIs

iwcfg_set_essid (IPCFG_default_enet_device, essid);
This API sets the network to which the GS1011 module has to connect. The first argument is the device number (1 for the GS1011 module) and the second argument is the string that contains the network name to which the module will connect.

#define DEMOCFG_SECURITY "none"

iwcfg_set_sec_type (IPCFG_default_enet_device,DEMOCFG_SECURITY);
This API sets the security type of the network to join as open.

iwcfg_set_mode (IPCFG_default_enet_device,DEMOCFG_NW_MODE);
This API triggers the GS1011 module to scan and connect to a network.

The above calls are sufficient for the GS1011 module to set up a Wireless link. If the network to join is security enabled, security key and security type have to be configured with iwcfg_set_passphrase and iwcfg_set_sec_type as follows (before iwcfg_set_mode).

#define DEMOCFGPSK "12345678"

iwcfg_set_passphrase(IPCFG_default_enet_device,(char_ptr)DEMOCFGPSK);

#define DEMOCFG_SECURITY "wpa2"

iwcfg_set_sec_type (IPCFG_default_enet_device,DEMOCFG_SECURITY);

The following are the example calls to set the transmit power, wireless retry count etc. These parameters need to be configured prior to making a call to iwcfg_set_mode.

Some of the iwconfig interfaces like set wps and set web provisioning parameters are currently not available in the RTCS. To include them, the RTCS can be re-built with the supplied iwconfig.c and iwconfig.h that contain these missing interface definitions. If the RTCS is not updated in this manner, then set_wps, set_webprovisioning calls have to be avoided by the application and the ENET_mediactl interface can directly be used.

To put the GS1011 module into power save mode, the following iwconfig interface needs to be invoked as shown below.

iwcfg_set_power (

```
        IPCFG_default_enet_device,
    0,
    FALSE
);
```

To bring the GS1011 module out of power save mode, use the same iwconfig interface as shown below.

```
iwcfg_set_power (
        IPCFG_default_enet_device,
    0,
    TRUE
);
```

Similarly to scan all channels and get the scan information as shown below. The scan results are getting printed on the serial console.

```
iwcfg_set_scan (
    IPCFG_default_enet_device,
    NULL
)
```

Similarly using the ENET_mediactl to scan all channels as shown below.

```
ENET_SCAN_LIST  param;
ENET_mediactl (ENET_handle,ENET_SET_MEDIACTL_SCAN,NULL);
ENET_mediactl (ENET_handle,ENET_GET_MEDIACTL_SCAN,&param);
```

10 Appendix B: Interface Diagram

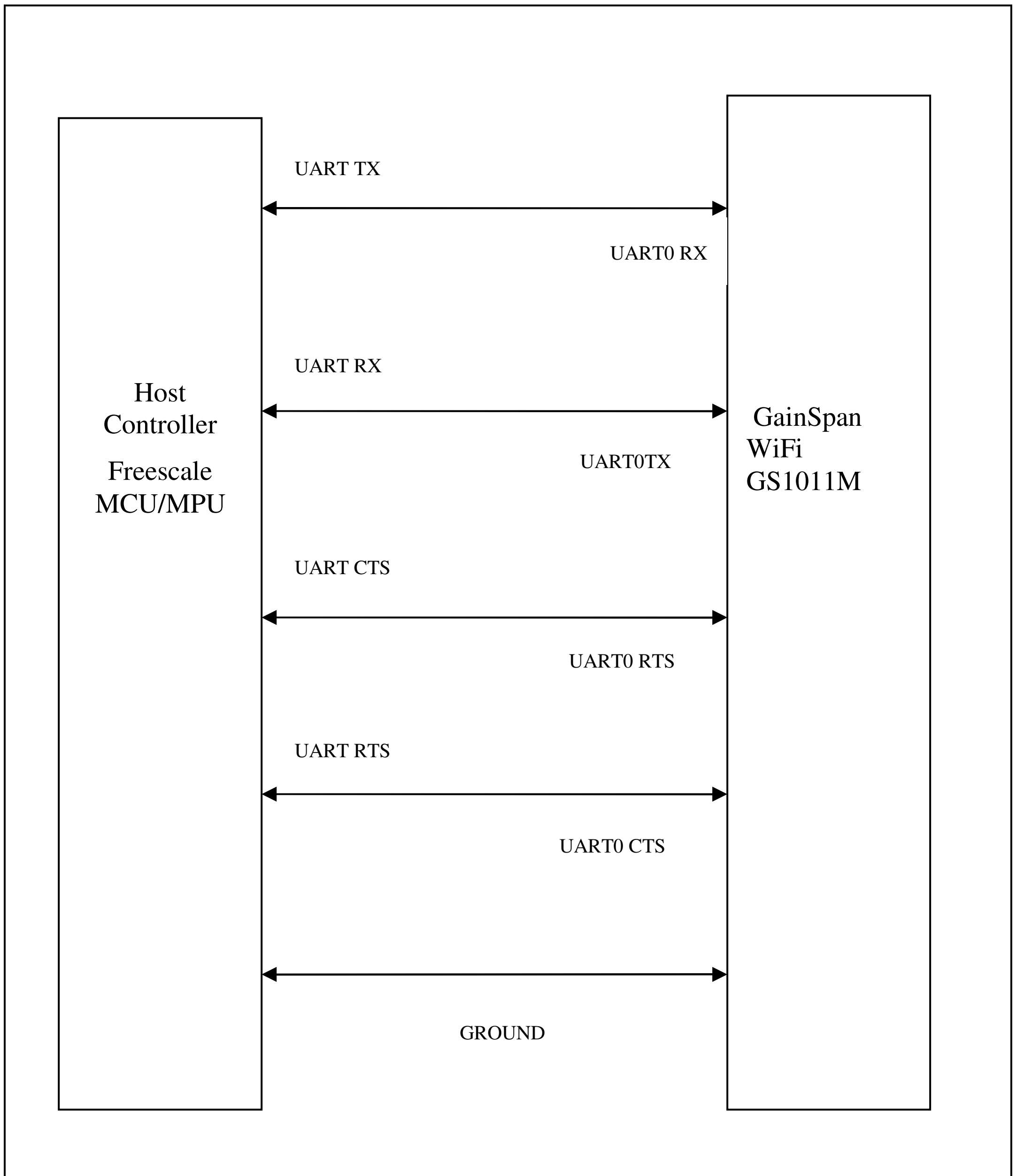


Fig 1: Hardware Interface diagram -UART

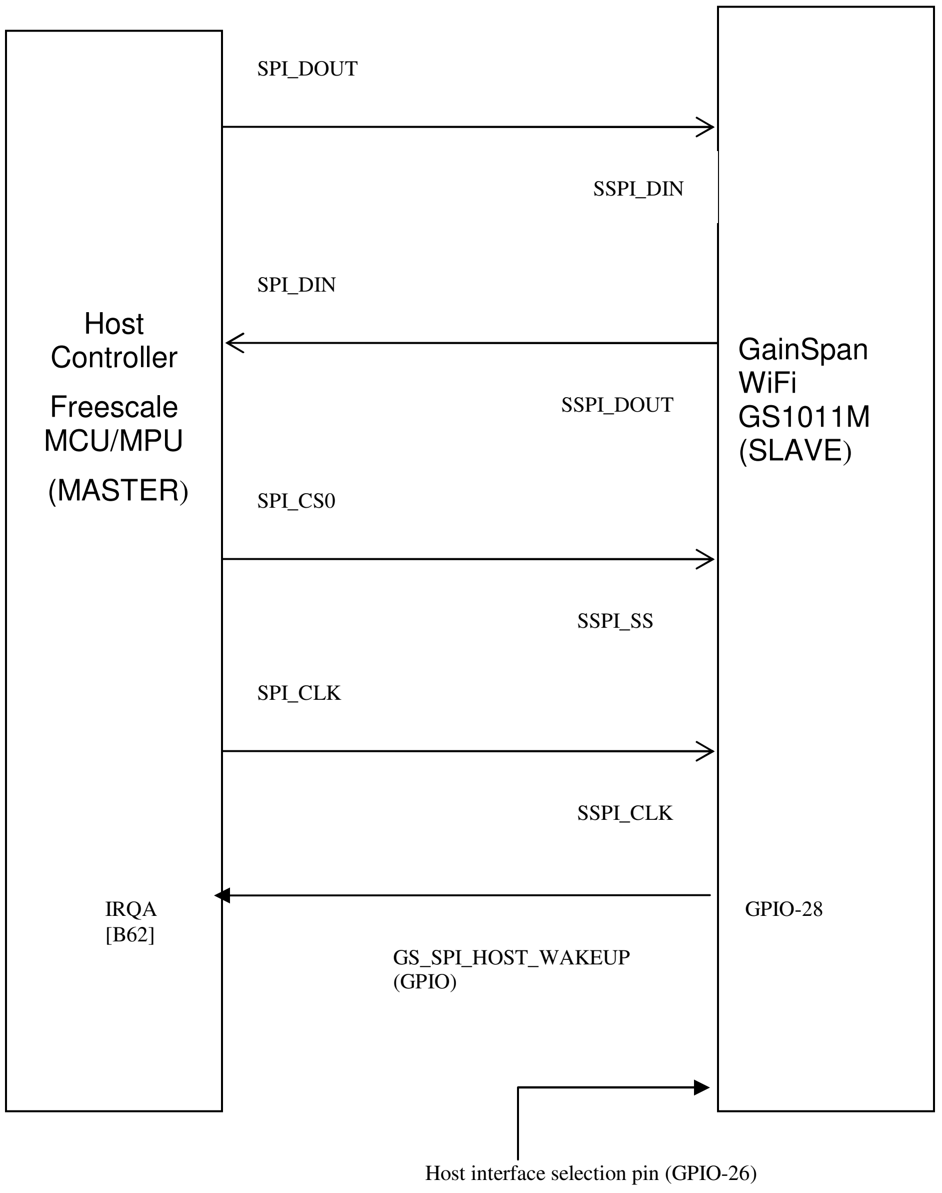


Fig 2: Hardware Interface diagram - SPI

11 Appendix C:

11.1 Data transfer over SPI interface:

In the case of SPI interface, the GS101X node acts as slave and will communicate to master SPI controller. By default, SPI interface supports Motorola protocol with clock polarity 0 and clock phase 0.
Since SPI data transfer works in full duplex mode, it’s required to make use of special octet to indicate idle data. Similarly if host controller is sending data at higher rate flow control mechanism is required. In order to differentiate these special control codes (such as idle pattern, flow control codes and other control octets) from user data, byte stuffing mechanism is incorporated.

SPI transmit data handling procedure:
The SPI data transfer layer makes use of an octet (or byte) stuffing procedure. The Control Escape octet is defined as binary 11111011 (hexadecimal 0xFB), most significant bit first. Each special control pattern is replaced by a two octet sequences consisting of the Control Escape octet followed by the original octet exclusive-or’d (XOR) with hexadecimal 0x20. Receiving implementations must correctly process all Control Escape sequences.

Escaped data is transmitted on the link as follows:

Pattern	Encoded as	Description
0xFD	0xFB 0xDD	Flow control XON
0xFA	0xFB 0xDA	Flow control XOFF
0x00	0xFB 0x20	Inactive link detection
0xFB	0xFB 0xDB	Control ESCAPE
0xF5	0xFB 0xD5	IDLE character
0xFF	0xFB 0xDF	Inactive link detection
0xF3	0xFB 0xD3	SPI link ready indication

One dedicated GPIO signal (GS_SPI _HOST_WAKEUP: GPIO#28) is available for data ready indications from Slave GS1011 node to Master Host controller. This GS_SPI _HOST_WAKEUP signal is asserted high during valid data transmission period, so that the host (master SPI) starts pulling out data by giving SPI clock and GS_SPI _HOST_WAKEUP signal is de-asserted once transmission is completed. Master host controller must provide clock as long as GS_SPI_HOST_WAKEUP signal is active.
Special character (0xF5) will be transmitted during idle period (if there is no more data to transmit) and must be dropped at the receiving Host.

SPI Receive data handling

Since byte stuffing is used, each Control Escape octet must be removed, and the next immediate octet is exclusive-or'd (XOR) with hexadecimal **0x20**.

If receive buffer is reached upper water mark, then ***XOFF*** character (0xFA) will be sent out informing the host to stop transmitting actual data. After receiving ***XOFF*** character host must stop transmitting actual data. Once the application starts processing received data and enough space available for further reception (reached lower water mark), ***XON*** (0xFD) will be transmitted. Once host receives ***XON***, then it can resume the valid data transmission.

Special IDLE characters (0xF5) will be dropped at receiver.

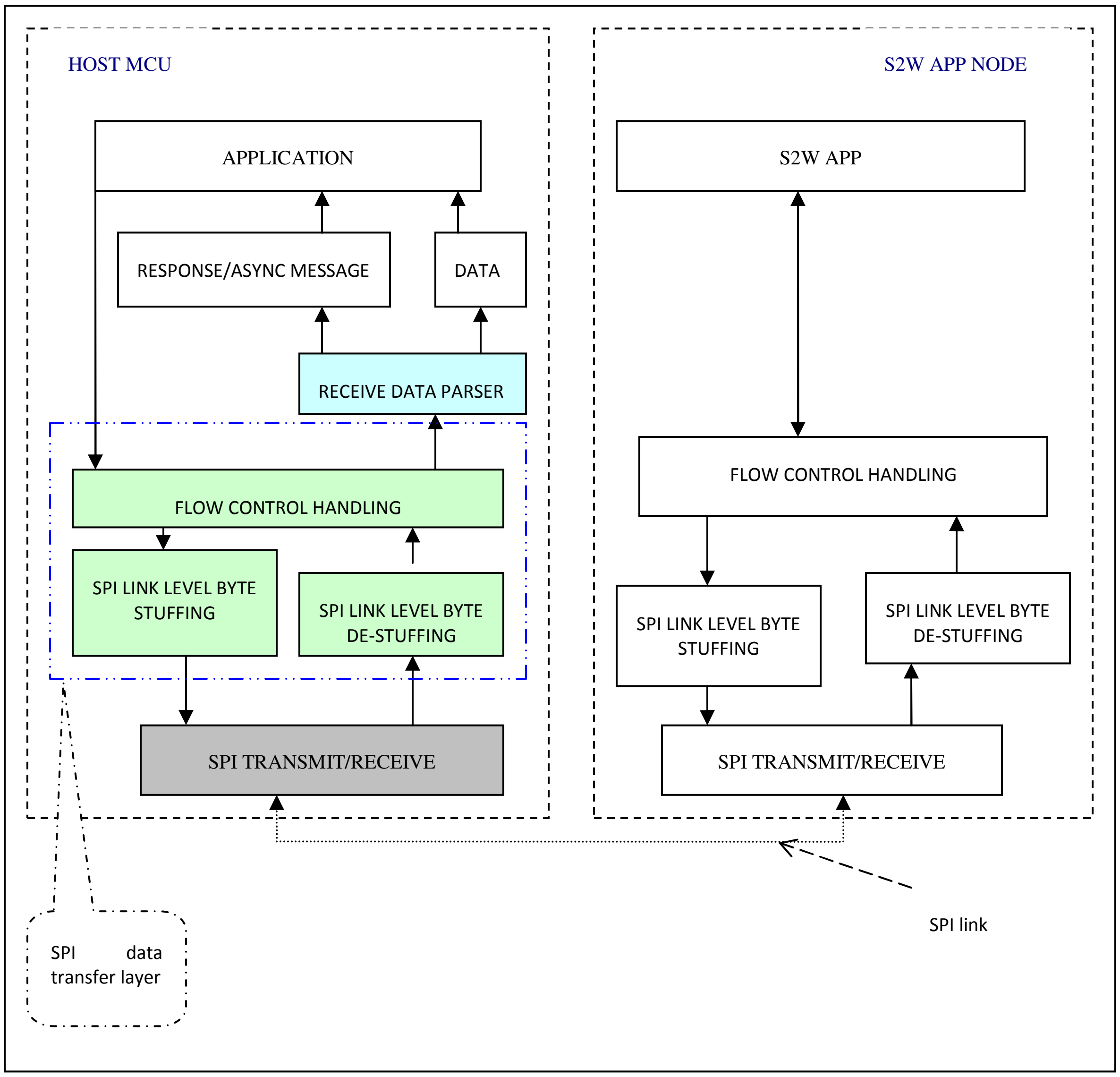


Fig: Data transfer over SPI interface

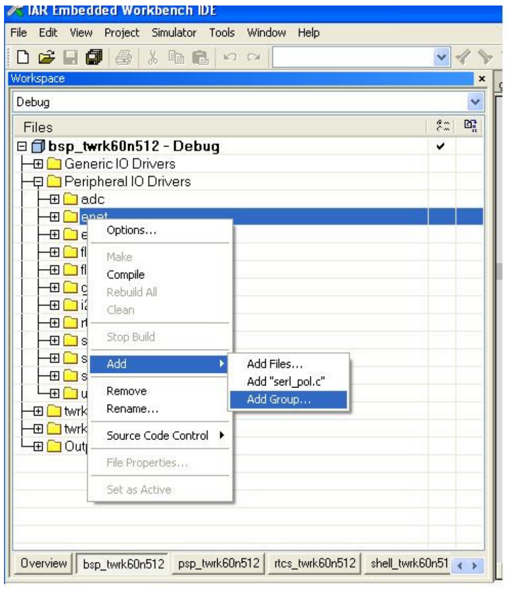
12 Adding GS Wi-Fi Driver file to IAR workbench

Adding driver files to BSP project:

1. Open the project file 'mqx\build\iar\bsp_twrk60n512.ewp'

2. Click on 'Peripheral I/O Drivers'

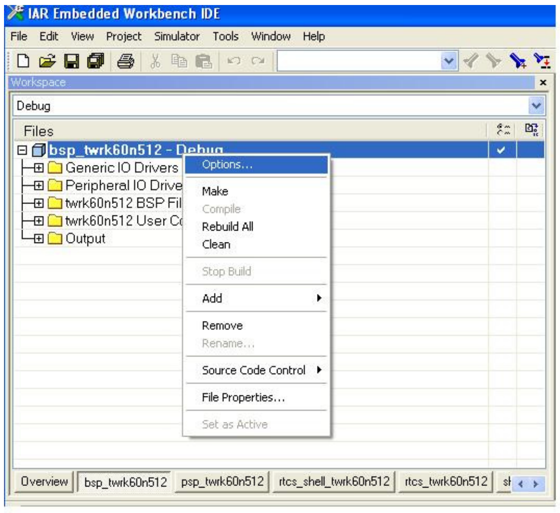
3. Right click on 'enet' and select 'add' ==> 'add group'



4. Type group name as 'gs'

5. Now Right click on 'gs' and select 'add' ==> 'add files' option.

6. Select all the files in the folder 'mqx\source\io\enet\gs'



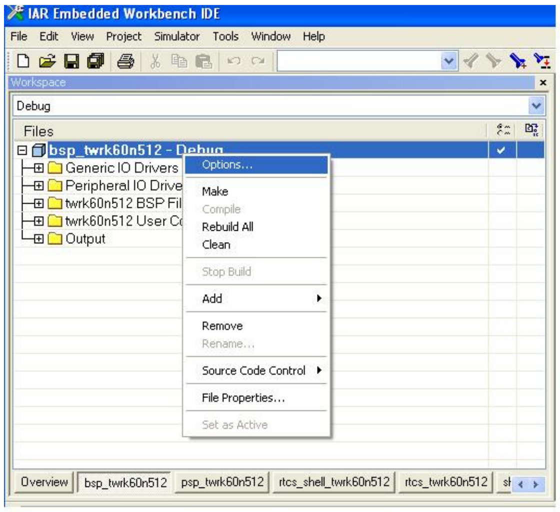
7. Similarly right click on 'phy' and select 'add' ==> 'add files' option.

8. Select 'phy_gs.c' and 'phy_gs.h' files to add

Now include header file path:

1. Open the project file 'mqx\build\iar\bsp_twrk60n512.ewp'

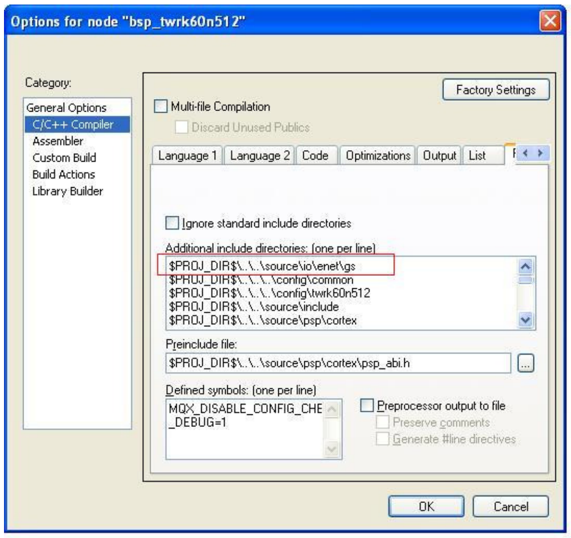
2. Right click on 'bsp_twrk60n512' and select 'options'



3. Click on 'C/C++ Compiler'

4. Click on 'Preprocessor'

5. Add '$PROJ_DIR$\..\..\source\io\enet\gs' path in Additional include directory window.



Ignore standard include directories

Additional include directories: (one per line)

```
$PROJ_DIR$\..\..\source\io\enet\gs
$PROJ_DIR$\..\..\..\config\common
$PROJ_DIR$\..\..\..\config\twrk60n512
$PROJ_DIR$\..\..\source\include
$PROJ_DIR$\..\..\source\psp\cortex
```

Preinclude file: $PROJ_DIR$\..\..\source\psp\cortex\psp_abi.h

Defined symbols: (one per line)

```
MQX_DISABLE_CONFIG_CHE
_DEBUG=1
```

Preprocessor output to file

Preserve comments

Generate #line directives

Debugger configuration for IAR work bench:

1. Open the main application project file (for example in case of http server application open the following project *~\rtcs\examples\httpsrv\iar\httsrv_twrk60n512.ewp'*)

2. Right click on project file *'httsrv_twrk60n512'* and select 'options'

3. Click on 'Debugger'

4. Select Driver as 'PE micro'