Application Note: AN00115

Building a Data Logger Application Note

Data logger continuously capture and record data from external sensors and instruments, and are an essential part of condition monitoring and preventative maintenance systems. Upon activation, data loggers are typically deployed and left unattended to measure and record information for the duration of monitoring period. This allows for a comprehensive, accurate picture of the environment conditions being monitored, such as temperature, relative humidity, production quantity monitoring, fuel level, energy consumption, pressure etc. XMOS technology is perfectly suited to these applications - offering future proof and reliable analog acquisition whilst offering the flexibility to interface to a huge variety of “Things”.

This application note shows a simple example that demonstrates the use of the XMOS ADC library to create a data logger using various analogue sensors on an XMOS multicore microcontroller.

The code associated with this application note provides an example of using the XMOS TCP/IP library as a framework for the creation of an ethernet based data logger.

Required tools and libraries

- Python on host workstation - Version 2.7.3
- xTIMEcomposer Tools - Version 13.1.0
- XMOS Ethernet library - Version 2.2.4
- XMOS TCP/IP library - Version 3.2.1
- XMOS OTP Function library - Version 1.0.0rc0
- XMOS xCORE-Analog Support library - Version 1.0.0rc0
- XMOS General utility: modules for developing XMOS devices
  - module_locks, module_logging and module_xassert - Version 1.0.3rc0

Required hardware

This application note is designed to run on an XMOS xCORE-Analog family (A-Series) device. The example code provided with the application has been implemented and tested on the xCORE-Analog sliceKIT 1V0 (XP-SKC-A16) core board using mixed signal sliceCARD 1V3 (XA-SK-MIXED SIGNAL) and ethernet sliceCARD 1V1 (XA-SK-E100). There is no dependancy on this core board - it can be modified to run on any (XMOS) development board which has the **Multichannel ADC** option.

Prerequisites

- This document assumes familarity with the XMOS xCORE architecture, the Ethernet standards IEEE 802.3u (MII), the XMOS tool chain and the xC language. Documentation related to these aspects which are not specific to this application note are linked to in the references appendix.
- For a description of XMOS related terms found in this document please see the XMOS Glossary[1].
- For an overview of Ethernet Layer 2 MAC library, please see the *Layer 2 Ethernet MAC*[2].
- For an overview of Ethernet TCP/IP library, refer *Ethernet TCP/IP component programming guide*[3].
- For an overview of Analog support library, check *xCORE-Analog Support Library*[4].

[1] http://www.xmos.com/published/glossary
[2] https://www.xmos.com/download/public/module_ethernet-README(2.2.4rc0.a).pdf
[3] https://www.xmos.com/download/public/Ethernet-TCP-IP-Component-Programming-Guide-(documentation)(3.2.1rc1.a).pdf
[4] https://www.xmos.com/download/public/xCORE-Analog-Support-Library-(documentation)(1.0.0rc0.a).pdf

1 Overview

1.1 Introduction

This application note describes how to build a data logger using the XMOS xCORE-Analog device. This implementation uses four logical cores and logs the sensor information periodically. The sensor data which are acquired from LDR (Light Dependent Resistor), Temperature sensor and Joystick are displayed on the python script console as well as logged on to an *datalogger.log* file.

The standard XMOS TCP/IP component is used to send the sensor information to the host workstation.

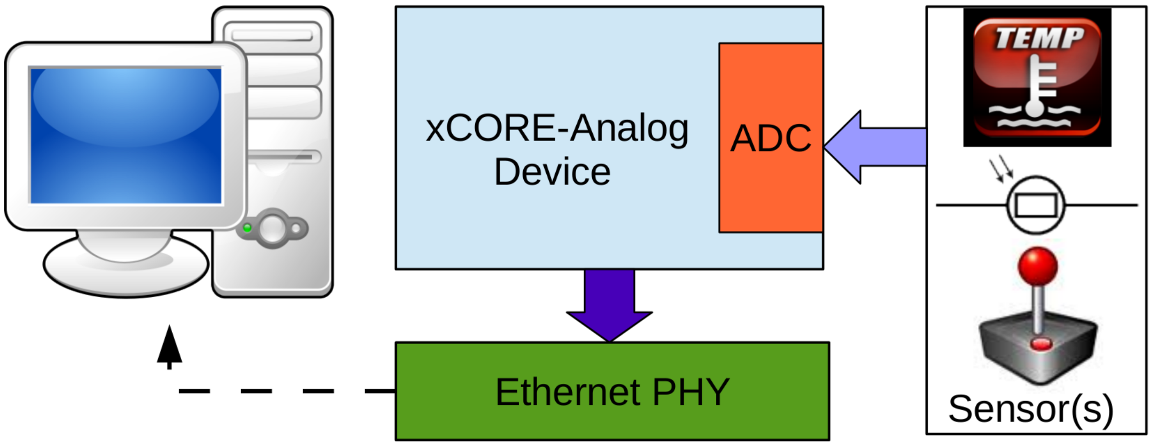


Figure 1: XMOS Data Logger

1.2 Block Diagram

Block diagram of XMOS data logger showing cores which are involved for this application

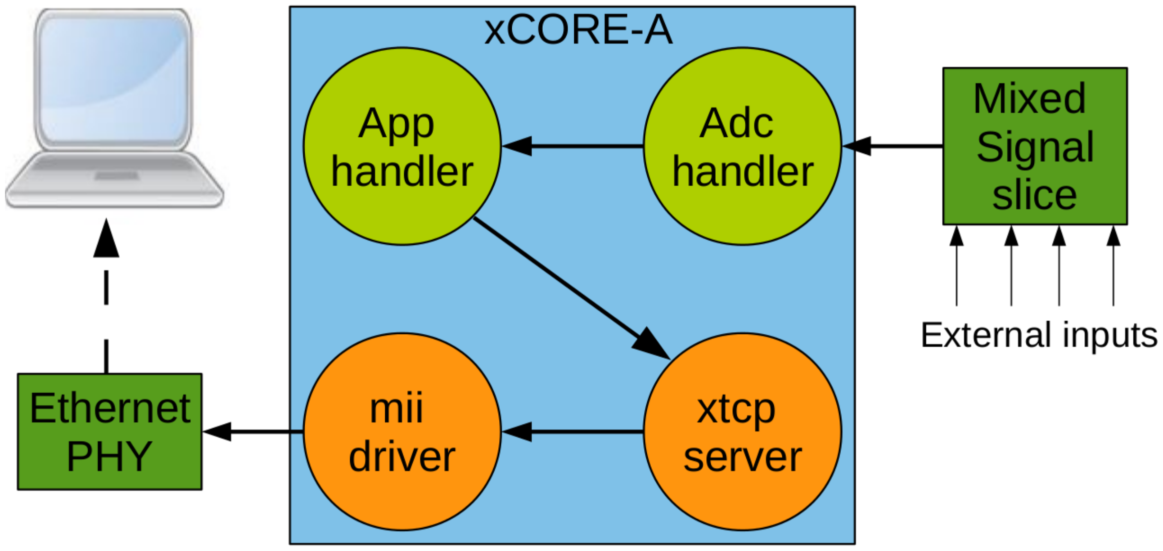


Figure 2: Block diagram of XMOS data logger

2 XMOS data logger application note

The demo in this note uses the XMOS XTCP and analog tile support libraries and shows a simple program that logs the analogue data from the in-built sensors/interface available in Mixed Signal sliceCARD.

To start constructing the data logger, you need to add *module_analog_tile* and *module_xtcp* to your makefile:

```
USED_MODULES = module_analog_tile module_xtcp
```

You can then access the ADC library functions in your source code via the **analog_tile_support.h** header file:

```
#include "analog_tile_support.h"
```

and XTCP library functions via the **xtcp.h** header file:

```
#include "xtcp.h"
```

Note: You need to create ethernet_board_conf.h header file to configure the ethernet port based on the slot it is connected.

2.1 ADC trigger port

There is a dedicated 32-bit port used to capture the ADC samples.

```
//::port to trigger ADC
on tile[0]: port trigger_port = PORT_ADC_TRIGGER;
//::
```

2.2 Setting the IP Address

Set an IP address that is routable in the network that the application is to be tested on.

```
//::client_ipconfig
xtcp_ipconfig_t client_ipconfig = {
  {169, 254, 202, 190},
  {255, 255, 0, 0},
  {0, 0, 0, 0}
};
//::client

//::server_config
server_config_t server_config = {
  {169, 254, 202, 189},
   80, 80 };
//::server_ip
```

2.3 ADC configuration

Configuring the ADC module requires the input channels, bit resolution, samples per packet and calibration mode to be defined.

1. In-built sensors/interface on the Mixed Signal sliceCARD which are connected to channels ADC[0:3] are enabled.
2. Resoultion is set to 8-bit

3. Samples per packet i.e. number of channels to sample is set to four.
4. Calibration mode is disabled in order to take the external voltage as input.

```
//::adc_config initialise
at_adc_config_t adc_config = { {0, 0, 0, 0, 0, 0, 0, 0}, 0, 0, 0};
//::adc_config

//::enabling input ADC channels
adc_config.input_enable[0] = 1; //Input 0 is LDR
adc_config.input_enable[1] = 1; //Input 1 is thermistor
adc_config.input_enable[2] = 1; //Input 2 is horizontal axis of the joystick
adc_config.input_enable[3] = 1; //Input 3 is vertical axis of the joystick
//::input_channels

//::config sampling rate and mode
adc_config.bits_per_sample = ADC_8_BPS;
adc_config.samples_per_packet = 4; //Allow samples to be sent in one hit
adc_config.calibration_mode = 0;
//::adc_sampling

//::enable_adc
at_adc_enable(analog_tile, c_adc, trigger_port, adc_config);
at_adc_trigger_packet(trigger_port, adc_config); //Fire the ADC!
//::trigger_adc
```

2.4 Ethernet configuration

Configuring the ethernet client requires,

1. Setting-up the server parameters
2. Initializing the client and waiting till connection is UP.
3. Once client is UP, it connets the server and waiting till NEW_CONNECTION is available.

```
// Set socket server parameters
sock_client_set_server_config(server_config);
// Initialize socket client
sock_client_init(c_xtcp);
// Connect to server
sock_client_connect_to_server(c_xtcp);
```

2.5 ADC data packet readout

By calling the function **at_adc_read_packet(c_adc, adc_config, data)** user can readout the ADC count values.

2.6 The application main() function

Below is the source code for the main function of this application, which is taken from the source file **main.xc**

1. Enables ethernet xtcp server.
2. Enables application handler to initialize ethernet client, wait for sensor data and send data to ethernet server.
3. Enables ADC handler to initialize ADC, acquire ADC data packets and send sensor data to application handler.

```
int main(void) {
    chan c_xtcp[1], c_adc;
    streaming chan c_logger;

    par {
        on ETHERNET_DEFAULT_TILE: ethernet_xtcp_server(xtcp_ports, client_ipconfig, c_xtcp, 1);
        on tile[0]: app_handler(c_xtcp[0], server_config, c_logger);
        on tile[0]: adc_handler(c_logger, c_adc, trigger_port);
        xs1_a_adc_service(c_adc);
    }

    return 0;
}
```

2.7 The application handler function

This function (core) handles the sensor data transfer between ADC handler and socket client.

1. Sets the server parameters.
2. Initializes the client.
3. Connects to the server.
4. Sends logger notifier information to the server.
5. Kicks-off adc handler to start its conversion
6. Gets sensor data from ADC handler and converts into readable string
7. Sends the logger(sensor) data information to server.

This *app_handler()* acts as a ethernet client which continues to run between step 6 and 7 later.

Below is the source code for the application function, which is taken from the source file **app_handler.xc**

```
void app_handler(chanend c_xtcp, server_config_t server_config, streaming chanend c_logger) {

  sensor_data_t sensor_data;

  // Set socket server parameters
  sock_client_set_server_config(server_config);
  // Initialize socket client
  sock_client_init(c_xtcp);
  // Connect to server
  sock_client_connect_to_server(c_xtcp);

  // Send notification to begin recording sensor data
  sock_client_send_data(c_xtcp, logger_notifier);

  // application is ready. The app_handler can now begin to record data.
  c_logger <: 1;

  while(1) {
    c_logger :> sensor_data;
    convert_to_string(sensor_data);
    sock_client_send_data(c_xtcp, logger_data);
  } // while
}
```

2.8 The ADC handler function

This function (core) reads out the ADC data from Multichannel Analog tile as packets

1. Enables the required analog channels
2. Configures ADC bit resolution and number of ADC channels per conversion
3. Enables the ADC and starts conversion.
4. Waits for the application handler to send kick-off notification.
5. At every time period based on *ADC_TRIGGER_PERIOD* ADC starts it conversion.

6. Readout data are stored in structure buffer
7. Buffered data is sent to application handler periodically based on *LOGGER_TRIGGER_PERIOD*. *(Note: while running on CONFIG_CHANGE_STATE option, buffered datas are send to application handler immediately when there is any change in state).*

This *adc_handler()* continues to run between step 5 to 7 later.

Below is the source code which shows how the ADC gets triggered periodically and how the sample data are readout. This is taken from the source file **adc_handler.xc**

```
void adc_handler(streaming chanend c_logger, chanend c_adc, port trigger_port)
{
  unsigned data[4];          // Array for storing ADC results
  timer adc_trigger_timer;
  unsigned adc_trigger_time;
  sensor_data_t sensor_data_l;

#ifdef CONFIG_PERIODIC
  timer periodic_timer;
  unsigned periodic_interval;
#endif

  sensor_data.ldr = 0;
  sensor_data.joystick_x = 0;
  sensor_data.joystick_y = 0;
  sensor_data.temperature = 0;

  //::adc_config initialise
  at_adc_config_t adc_config = { {0, 0, 0, 0, 0, 0, 0, 0}, 0, 0, 0};
  //::adc_config

  //::enabling input ADC channels
  adc_config.input_enable[0] = 1; //Input 0 is LDR
  adc_config.input_enable[1] = 1; //Input 1 is thermistor
  adc_config.input_enable[2] = 1; //Input 2 is horizontal axis of the joystick
  adc_config.input_enable[3] = 1; //Input 3 is vertical axis of the joystick
  //::input_channels

  //::config sampling rate and mode
  adc_config.bits_per_sample = ADC_8_BPS;
  adc_config.samples_per_packet = 4; //Allow samples to be sent in one hit
  adc_config.calibration_mode = 0;
  //::adc_sampling

  //::enable_adc
  at_adc_enable(analog_tile, c_adc, trigger_port, adc_config);
  at_adc_trigger_packet(trigger_port, adc_config); //Fire the ADC!
  //::trigger_adc

  // Wait till the app handler is ready
  c_logger :> int _;

  adc_trigger_timer :> adc_trigger_time;         //Set timer for first loop tick
  adc_trigger_time += ADC_TRIGGER_PERIOD;

#ifdef CONFIG_PERIODIC
  periodic_timer :> periodic_interval;
  periodic_interval += LOGGER_TRIGGER_PERIOD;
#endif

  while(1)
  {
    select
    {
#pragma ordered
      case adc_trigger_timer when timerafter(adc_trigger_time) :> adc_trigger_time:
      {
        at_adc_trigger_packet(trigger_port, adc_config);    //Trigger ADC
        adc_trigger_time += ADC_TRIGGER_PERIOD;
        break;
      } // case loop_timer to trigger ADC

      case at_adc_read_packet(c_adc, adc_config, data): //if data ready to be read from ADC
      {
        unsigned char flag_ldr = 0, flag_temp = 0, flag_jx = 0, flag_jy = 0; // clear all flags

        flag_ldr = value_beyond_limits(data[0], sensor_data_l.ldr, 1);
        flag_temp = value_beyond_limits(celsius_temperature(data[1]), sensor_data_l.temperature, 1);
        flag_jx = value_beyond_limits(data[2], sensor_data_l.joystick_x, 1);
        flag_jy = value_beyond_limits(data[3], sensor_data_l.joystick_y, 1);

        sensor_data_l.ldr = data[0];           //ADC0 value
        sensor_data_l.temperature = celsius_temperature(data[1]); //ADC1 value
        sensor_data_l.joystick_x  = data[2];   //ADC2 value
        sensor_data_l.joystick_y  = data[3];   //ADC3 value
        sensor_data = sensor_data_l;

#ifdef CONFIG_CHANGE_STATE // send ADC data to logger, if there is any change in state
        if(flag_ldr || flag_temp || flag_jy || flag_jy)
        {
           c_logger <: sensor_data;
        }
#endif
        break;
      } // case at_adc_read_packet

#ifdef CONFIG_PERIODIC // send ADC data to logger, periodically
      case periodic_timer when timerafter(periodic_interval) :> void:
      {
        c_logger <: sensor_data;
        periodic_timer :> periodic_interval;
        periodic_interval += LOGGER_TRIGGER_PERIOD;
        break;
      } // case loop_timer to trigger logger
#endif
    } // select
  } // while(1)
}
```

APPENDIX A - Demo Hardware Setup

- To run the demo, connect the XTAG-2 USB debug adapter to the sliceKIT via the supplied adaptor board
- Connect the XTAG-2 to the host PC (using USB extension cable if desired)
- Connect the ethernet sliceCARD to the **SQUARE** slot of the sliceKIT. Then, connect the slice to the host PC or to the network switch using an ethernet cable.
- Connect the mixed signal sliceCARD to the **ANALOG** slot *(marked as "A")* of the sliceKIT. Connect pins 1 and 2 of J7 to use LDR as one of the ADC input.
- On the xCORE-A series sliceKIT ensure that the xCONNECT LINK (xSCOPE) switch is set to ON, as per the image, to allow xSCOPE to function.
- *Note: The MIXED SIGNAL sliceCARD uses ADC channels [0:3] for the in-built sensors/interfaces like LDR, Temperature sensor and Joystick available on the sliceCARD. User can provide external inputs on ADC channels [4:7] using J2 jumper.*

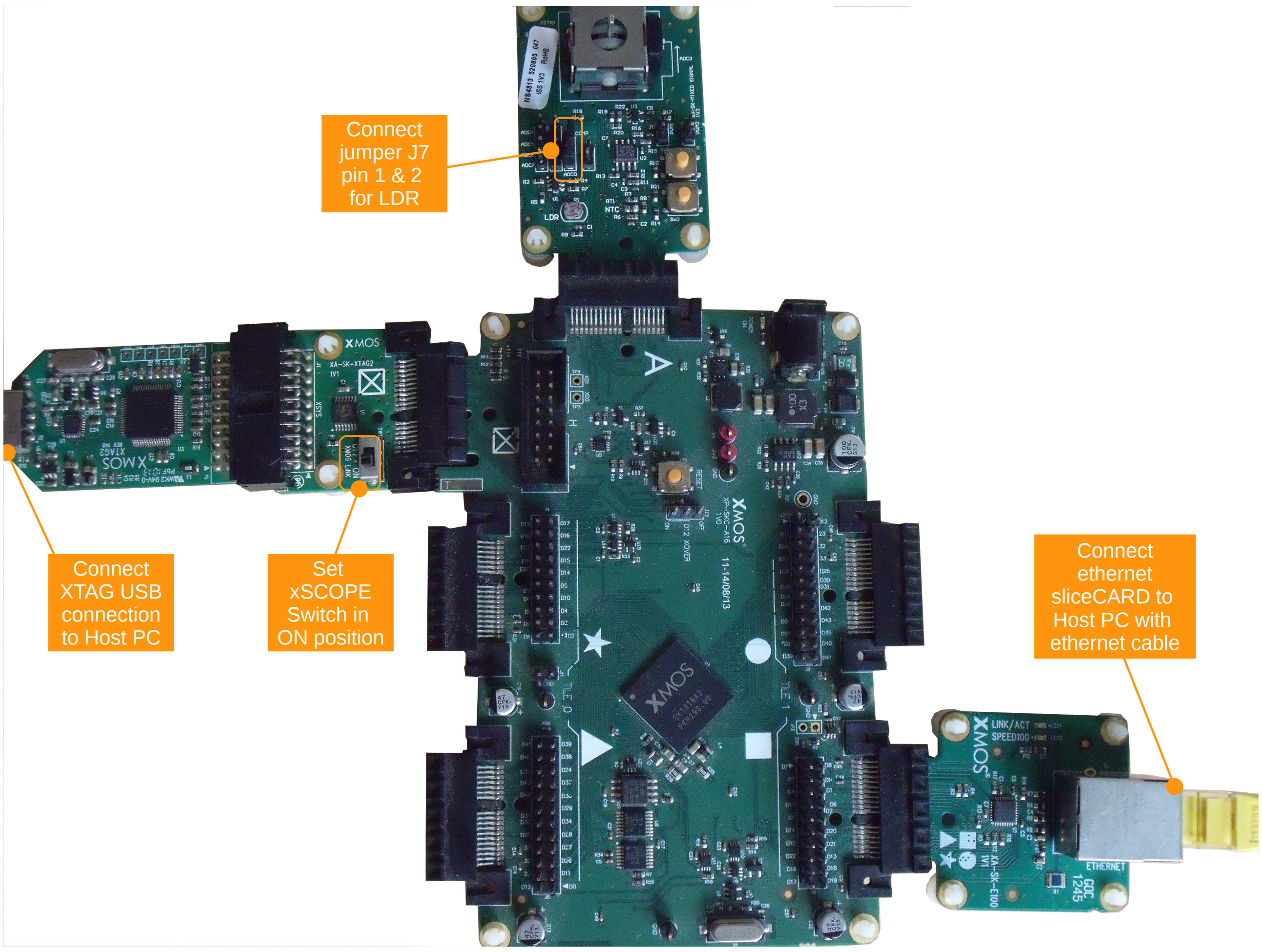
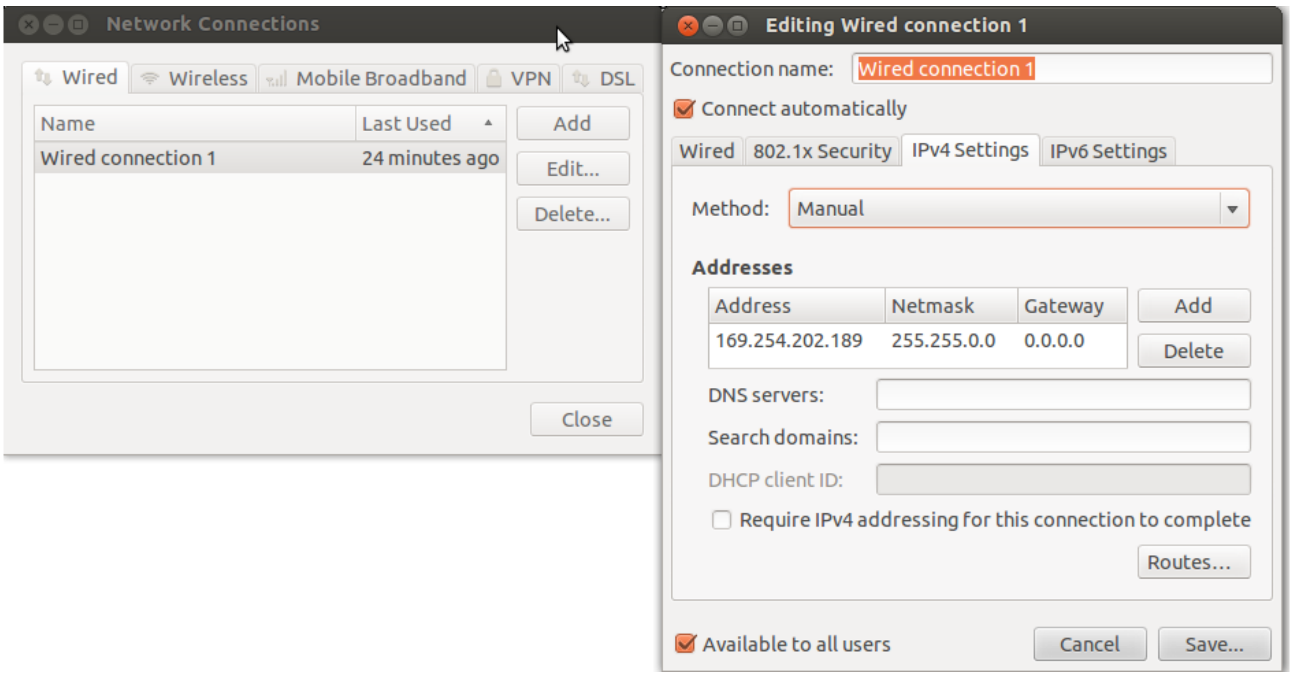

Connect jumper J7 pin 1 & 2 for LDR
Connect XTAG USB connection to Host PC
Set xSCOPE Switch in ON position
Connect ethernet sliceCARD to Host PC with ethernet cable

Figure 3: Hardware Setup for XMOS Data Logger

A.1 Host computer setup

- Required a host workstation with Python (2.7.3). Get Python from python.org[5].
- Configure the wired connection IPv4 settings to use a static IP address and netmask as follows. *IP address = 169.254.202.189 and Netmask = 255.255.0.0*
- *Note: The IP address which is configured above should be the same in the code (server_config).*[§2.2]
  - For Mac: Navigate to *System Preferences -> Network -> Ethernet -> Configure IPv4 -> Manually* and provide the IP address.
  - For Windows: Navigate to *Start -> Control Panel -> Network and Sharing Center -> Change Adapter Settings* (on the left pane)
    - ⋆ Double click on **Local Area Connection**
    - ⋆ Double click on **Internet Protocol Version 4**
    - ⋆ Select the option **Use the following IP address**
    - ⋆ Provide the IP address and Subnet mas (gateway can be blank) and Click **OK**
  - For Linux(Ubuntu): Navigate to *System Settings -> Network -> Wired ->* Edit *Wired Connection -> IPv4 Settings -> Manually* and provide the IP address in the space below it.



Name	Last Used
Wired connection 1	24 minutes ago

Connection name: Wired connection 1

☑ Connect automatically

Method: Manual

Addresses

Address	Netmask	Gateway
169.254.202.189	255.255.0.0	0.0.0.0

DNS servers:

Search domains:

DHCP client ID:

☐ Require IPv4 addressing for this connection to complete

☑ Available to all users

Figure 4: IP address setting in Linux host machine

[5] https://www.python.org/

APPENDIX B - Launching the demo device

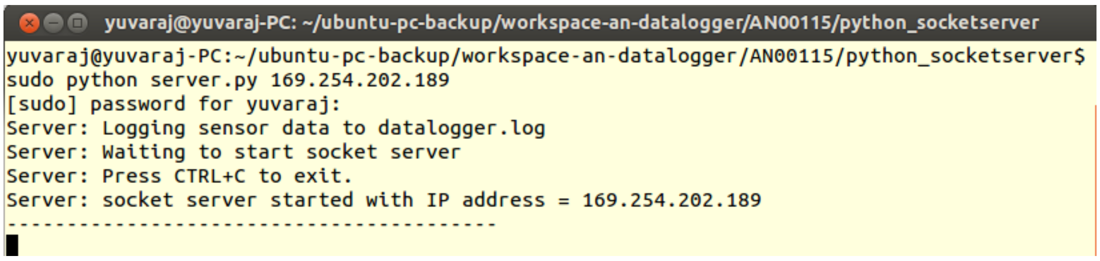
B.1 Run the Python script

- On the *Terminal* or *Command prompt* navigate to *AN00115/python_socketserver* folder.
- Run the Python script with the socket server IP address (*Note: Administrator privileges may be required to run server.py*).
  - For Windows: start command prompt as an administrator and then execute the Python script. The scripts may be executed by invoking Python from its installation path. Usually, Python will be installed in **C:\\.**:

    ```
    c:\Python2\python.exe server.py 169.254.202.189
    ```

  - For Mac/Linux: run the script with *sudo*:

    ```
    sudo python server.py 169.254.202.189
    ```



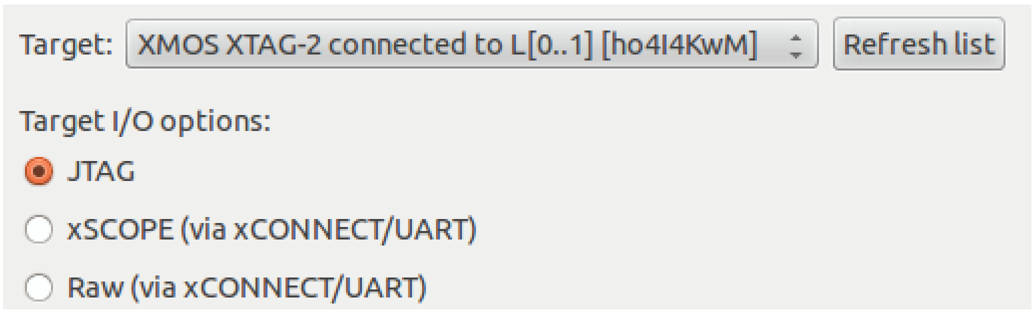
```
yuvaraj@yuvaraj-PC: ~/ubuntu-pc-backup/workspace-an-datalogger/AN00115/python_socketserver
yuvaraj@yuvaraj-PC:~/ubuntu-pc-backup/workspace-an-datalogger/AN00115/python_socketserver$
sudo python server.py 169.254.202.189
[sudo] password for yuvaraj:
Server: Logging sensor data to datalogger.log
Server: Waiting to start socket server
Server: Press CTRL+C to exit.
Server: socket server started with IP address = 169.254.202.189
----------------------------------------
```

Figure 5: Data logger Python script launch

B.2 Run the data logger application

Once the application source code is imported into the tools you can build the project which will generate the binary file required to run the demo application. Once the application has been built you need to download the application code onto the xCORE-A sliceKIT. Here you use the tools to load the application over JTAG onto the xCORE device.

- Select **Run Configuration.**
- In **Main** menu, enable **JTAG** in Target I/O options.
- In **XScope** menu, enable **Disabled**
- Click **Apply** and then **Run.**



Target: XMOS XTAG-2 connected to L[0..1] [ho4I4KwM]

Target I/O options:

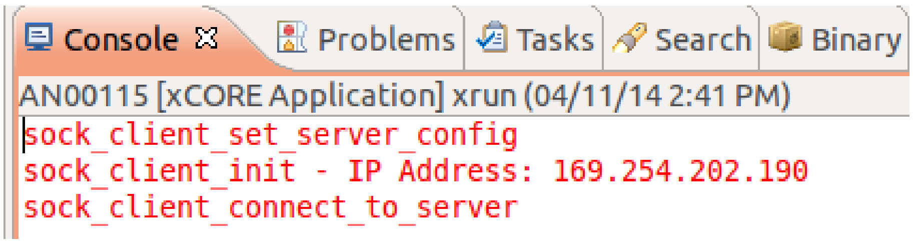
[✓ JTAG]

xSCOPE (via xCONNECT/UART)

Raw (via xCONNECT/UART)

Figure 6: xTIMEcomposer Target I/O configuration

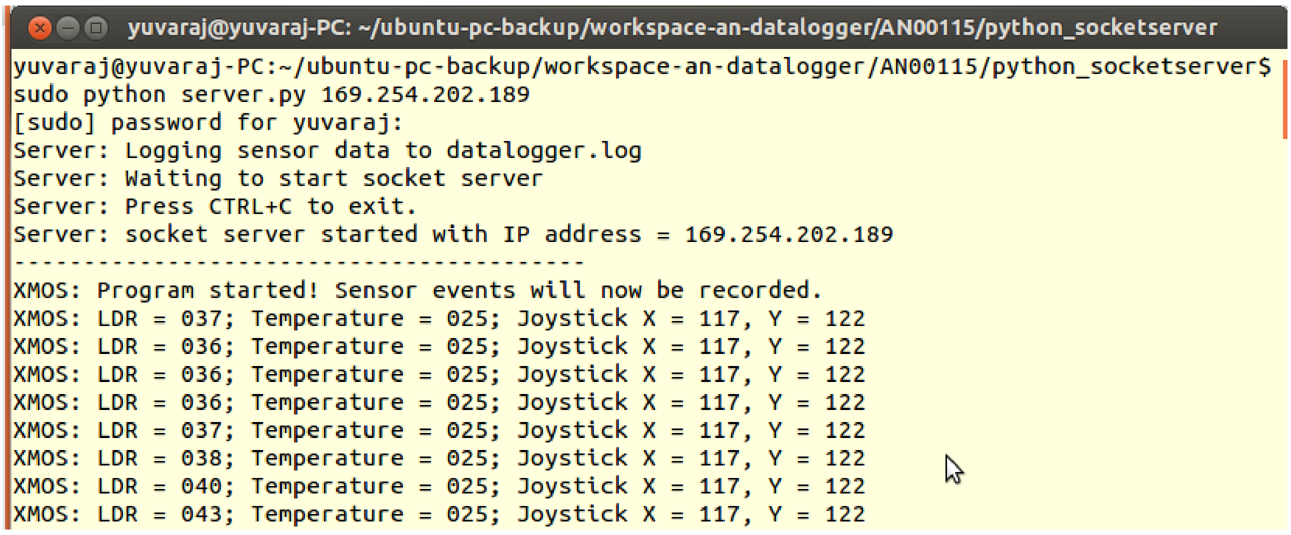
When the processor has finished booting you will see the below text in the xTIMEcomposer console window. The message **sock_client_connect_to_server** ensures that client & server got connected to each other successfully.



```
AN00115 [xCORE Application] xrun (04/11/14 2:41 PM)
sock_client_set_server_config
sock_client_init - IP Address: 169.254.202.190
sock_client_connect_to_server
```

Figure 7: xTIMEcomposer console

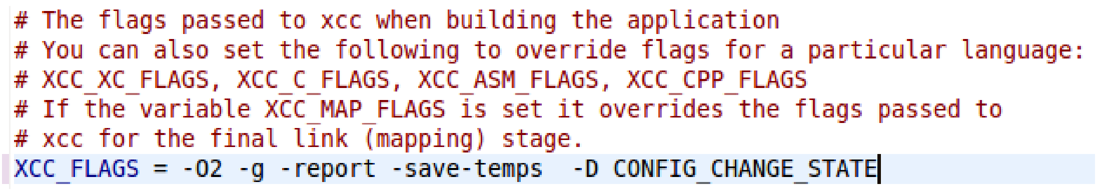
On the successful connection, data logger client will start sending the sensor data to the server (python script) periodically.



```
yuvaraj@yuvaraj-PC: ~/ubuntu-pc-backup/workspace-an-datalogger/AN00115/python_socketserver
yuvaraj@yuvaraj-PC:~/ubuntu-pc-backup/workspace-an-datalogger/AN00115/python_socketserver$
sudo python server.py 169.254.202.189
[sudo] password for yuvaraj:
Server: Logging sensor data to datalogger.log
Server: Waiting to start socket server
Server: Press CTRL+C to exit.
Server: socket server started with IP address = 169.254.202.189
-------------------------------------------
XMOS: Program started! Sensor events will now be recorded.
XMOS: LDR = 037; Temperature = 025; Joystick X = 117, Y = 122
XMOS: LDR = 036; Temperature = 025; Joystick X = 117, Y = 122
XMOS: LDR = 036; Temperature = 025; Joystick X = 117, Y = 122
XMOS: LDR = 036; Temperature = 025; Joystick X = 117, Y = 122
XMOS: LDR = 037; Temperature = 025; Joystick X = 117, Y = 122
XMOS: LDR = 038; Temperature = 025; Joystick X = 117, Y = 122
XMOS: LDR = 040; Temperature = 025; Joystick X = 117, Y = 122
XMOS: LDR = 043; Temperature = 025; Joystick X = 117, Y = 122
```

Figure 8: Python console showing the results

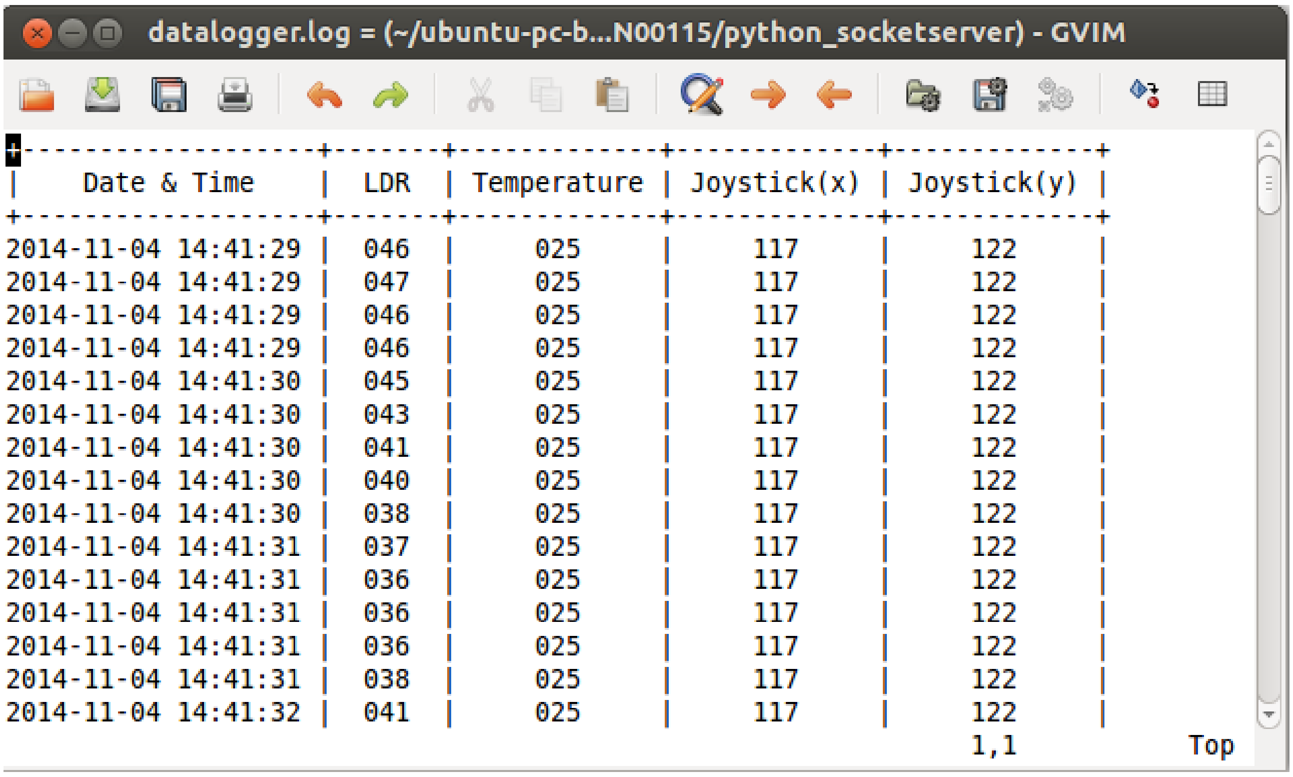
User can change the logging mechanism to *change in state* instead of *periodic* by changing the macro in **Makefile.** This will log only when there is any changes from the previous value to the present value on the sensors/interface (atleast one). *(For periodic logging use: CONFIG_PERIODIC, for change in state logging use: CONFIG_CHANGE_STATE)*



```
# The flags passed to xcc when building the application
# You can also set the following to override flags for a particular language:
# XCC_XC_FLAGS, XCC_C_FLAGS, XCC_ASM_FLAGS, XCC_CPP_FLAGS
# If the variable XCC_MAP_FLAGS is set it overrides the flags passed to
# xcc for the final link (mapping) stage.
XCC_FLAGS = -O2 -g -report -save-temps  -D CONFIG_CHANGE_STATE
```

Figure 9: Makefile changes for changing the logging mechanism

Logged information can be analysed from the *AN00115/python_socketserver/datalogger.log* file.



datalogger.log = (~/ubuntu-pc-b...N00115/python_socketserver) - GVIM

```
+-------------------+-------+-------------+-------------+-------------+
|     Date & Time     |  LDR  | Temperature | Joystick(x) | Joystick(y) |
+-------------------+-------+-------------+-------------+-------------+
2014-11-04 14:41:29 |  046  |     025     |     117     |     122     |
2014-11-04 14:41:29 |  047  |     025     |     117     |     122     |
2014-11-04 14:41:29 |  046  |     025     |     117     |     122     |
2014-11-04 14:41:29 |  046  |     025     |     117     |     122     |
2014-11-04 14:41:30 |  045  |     025     |     117     |     122     |
2014-11-04 14:41:30 |  043  |     025     |     117     |     122     |
2014-11-04 14:41:30 |  041  |     025     |     117     |     122     |
2014-11-04 14:41:30 |  040  |     025     |     117     |     122     |
2014-11-04 14:41:30 |  038  |     025     |     117     |     122     |
2014-11-04 14:41:31 |  037  |     025     |     117     |     122     |
2014-11-04 14:41:31 |  036  |     025     |     117     |     122     |
2014-11-04 14:41:31 |  036  |     025     |     117     |     122     |
2014-11-04 14:41:31 |  036  |     025     |     117     |     122     |
2014-11-04 14:41:31 |  038  |     025     |     117     |     122     |
2014-11-04 14:41:32 |  041  |     025     |     117     |     122     |
                                                          1,1         Top
```

Figure 10: Datalogger log file

APPENDIX C - FAQs

1. How to enable the external ADC input?

   User has to enable the respective *input_enable* in *adc_config* structure.:

```
//::enabling input ADC channels
adc_config.input_enable[4] = 1; //Input 4 is External input 0
adc_config.input_enable[5] = 1; //Input 5 is External input 1
adc_config.input_enable[6] = 1; //Input 6 is External input 2
adc_config.input_enable[7] = 1; //Input 7 is External input 3
//::input_channels
```

2. What is the maximum number of channels can be enabled simultaneously?

   All the eight channels can be enabled, but the buffer depth available on ADC module is only **five** in order the get accurate data. If in case the user has requirement to readout all eight channels, ADC samples per packet can be configured to four *(adc_config.samples_per_packet = 4;)* and packet readout can be made twice to get the remaining four channels.

3. How to enable calibration mode?

   When the *adc_config.calibration_mode* is set to one, the ADCs will sample a 0.8V reference rather than the external voltage.

4. How to change the resolution?

   By changing the *adc_config.bits_per_sample*, user can configure the ADC resolution to 8-bit or 12-bit. Options:

```
ADC_8_BPS      /**< Samples will be truncated to 8 bits */
ADC_16_BPS     /**< Samples will be placed in the MSB 12 bits of the half word */
ADC_32_BPS     /**< Samples will be placed in the MSB 12 bits of the word */
```

5. How to draw graphs?

   You need to install **matplotlib**[6] plotting library and run the python script *graph.py* which is available in **python_socketserver/** folder. This script draws graph using the data available in *datalogger.log* file.[Figure 11]

[6] http://matplotlib.org/users/installing.html

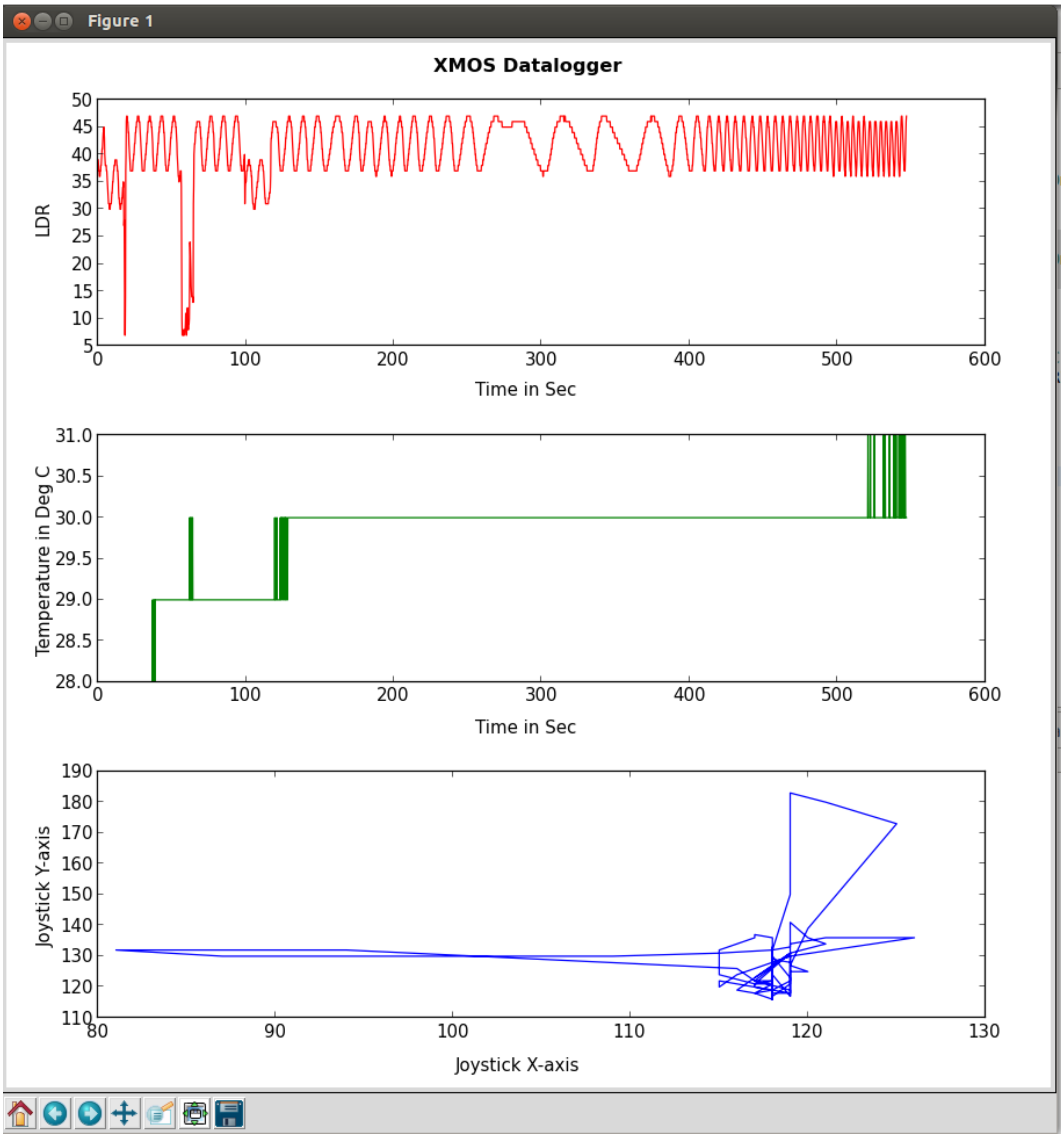


Figure 11: Graph drawn using Datalogger log file

APPENDIX D - References

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

XMOS TCP/IP Component[7]

XMOS Analog Component[8]

[7] https://www.xmos.com/download/public/Ethernet-TCP-IP-Component-Programming-Guide-(documentation)(3.2.1rc1.a).pdf

[8] https://www.xmos.com/download/public/xCORE-Analog-Support-Library-(documentation)(1.0.0rc0.a).pdf

APPENDIX E - Full Source code listing

E.1 Source code for main.xc

```
/**
 * The copyrights, all other intellectual and industrial
 * property rights are retained by XMOS and/or its licensors.
 * Terms and conditions covering the use of this code can
 * be found in the Xmos End User License Agreement.
 *
 * Copyright XMOS Ltd 2014
 *
 * In the case where this code is a modification of existing code
 * under a separate license, the separate license terms are shown
 * below. The modifications to the code are still covered by the
 * copyright notice above.
 *
 **/

#include <platform.h>
#include <xs1.h>
#include "analog_tile_support.h"
#include "adc_handler.h"
#include "app_handler.h"
#include "xtcp.h"
#include "ethernet_board_conf.h"
#include <stdio.h>
#include <string.h>

//::port to trigger ADC
on tile[0]: port trigger_port = PORT_ADC_TRIGGER;
//::

/* Interfaces port definitions
 * These intializers are taken from the ethernet_board_support.h header for
 * XMOS dev boards. If you are using a different board you will need to
 * supply explicit port structure intializers for these values  */
ethernet_xtcp_ports_t xtcp_ports =
  {    on ETHERNET_DEFAULT_TILE:  OTP_PORTS_INITIALIZER,
       ETHERNET_DEFAULT_SMI_INIT,
       ETHERNET_DEFAULT_MII_INIT_lite,
       ETHERNET_DEFAULT_RESET_INTERFACE_INIT};

//::client_ipconfig
xtcp_ipconfig_t client_ipconfig = {
  {169, 254, 202, 190},
  {255, 255, 0, 0},
  {0, 0, 0, 0}
};
//::client

//::server_config
server_config_t server_config = {
  {169, 254, 202, 189},
   80, 80 };
//::server_ip

int main(void) {
    chan c_xtcp[1], c_adc;
    streaming chan c_logger;

    par {
        on ETHERNET_DEFAULT_TILE: ethernet_xtcp_server(xtcp_ports, client_ipconfig, c_xtcp, 1);
        on tile[0]: app_handler(c_xtcp[0], server_config, c_logger);
        on tile[0]: adc_handler(c_logger, c_adc, trigger_port);
        xs1_a_adc_service(c_adc);
    }

    return 0;
}
```

E.2 Source code for app_handler.xc

```
/**
 * The copyrights, all other intellectual and industrial
 * property rights are retained by XMOS and/or its licensors.
 * Terms and conditions covering the use of this code can
 * be found in the Xmos End User License Agreement.
 *
 * Copyright XMOS Ltd 2014
 *
 * In the case where this code is a modification of existing code
 * under a separate license, the separate license terms are shown
 * below. The modifications to the code are still covered by the
 * copyright notice above.
 *
 **/
#include <stdio.h>
#include <string.h>
#include <print.h>
#include <time.h>
#include <platform.h>
#include "socket_client.h"
#include "app_handler.h"
#include "adc_handler.h"

char logger_notifier[] = " Program started! Sensor events will now be recorded.\n";
char logger_data[] = " LDR = bbb; Temperature = ttt; Joystick X = xxx, Y = yyy\n";

/*---------------------------------------------------------------------------
 convert_to_string
 ---------------------------------------------------------------------------*/
void convert_to_string(sensor_data_t sensor_data) {

    // conver to string
  logger_data[7] = sensor_data.ldr/100 + '0';
  logger_data[8] = (sensor_data.ldr%100)/10 + '0';
  logger_data[9] = sensor_data.ldr%10 + '0';
  if(sensor_data.temperature < 0) logger_data[26] = '-';
  else logger_data[26] = sensor_data.temperature/100 + '0';
  logger_data[27] = (sensor_data.temperature%100)/10 + '0';
  logger_data[28] = sensor_data.temperature%10 + '0';
  logger_data[44] = sensor_data.joystick_x/100 + '0';
  logger_data[45] = (sensor_data.joystick_x%100)/10 + '0';
  logger_data[46] = sensor_data.joystick_x%10 + '0';
  logger_data[53] = sensor_data.joystick_y/100 + '0';
  logger_data[54] = (sensor_data.joystick_y%100)/10 + '0';
  logger_data[55] = sensor_data.joystick_y%10 + '0';
}
/*---------------------------------------------------------------------------
 app_handler
 ---------------------------------------------------------------------------*/
void app_handler(chanend c_xtcp, server_config_t server_config, streaming chanend c_logger) {

  sensor_data_t sensor_data;

  // Set socket server parameters
  sock_client_set_server_config(server_config);
  // Initialize socket client
  sock_client_init(c_xtcp);
  // Connect to server
  sock_client_connect_to_server(c_xtcp);

  // Send notification to begin recording sensor data
  sock_client_send_data(c_xtcp, logger_notifier);

  // application is ready. The app_handler can now begin to record data.
  c_logger <: 1;

  while(1) {
    c_logger :> sensor_data;
    convert_to_string(sensor_data);
    sock_client_send_data(c_xtcp, logger_data);
  } // while
}
```

E.3 Source code for adc_handler.xc

```
#include "adc_handler.h"
#include "analog_tile_support.h"


#define ADC_TRIGGER_PERIOD      10000000 // 100ms for ADC trigger
#define TEMPERATURE_LUT_ENTRIES 16

#ifdef CONFIG_PERIODIC
#define LOGGER_TRIGGER_PERIOD   20000000 // 200ms for Log trigger
#endif
// sensor data
static sensor_data_t sensor_data;

// The temperature look-up table to convert ADC value from Thermistor to Celsius
static int TEMPERATURE_LUT[TEMPERATURE_LUT_ENTRIES][2] =
{
  {-10,211},{-5,202},{0,192},{5,180},
  {10,167},{15,154},{20,140},{25,126},
  {30,113},{35,100},{40,88},{45,77},
  {50,250},{55,230},{60,210}
};
/*---------------------------------------------------------------------------
 convert ADC value to temperature in Celsius
 ---------------------------------------------------------------------------*/
static int celsius_temperature(int adc_value)
{
  int i = 0, x1, y1, x2, y2, celsius = 0;

  while((adc_value < TEMPERATURE_LUT[i][1]) && (i < TEMPERATURE_LUT_ENTRIES)) i++;

  if (i != TEMPERATURE_LUT_ENTRIES)
  {
    x1 = TEMPERATURE_LUT[i-1][1];
    y1 = TEMPERATURE_LUT[i-1][0];
    x2 = TEMPERATURE_LUT[i][1];
    y2 = TEMPERATURE_LUT[i][0];
    celsius = y1 + (((adc_value - x1) * (y2 - y1)) / (x2 - x1));
  }
  return celsius;
}

#define YES   1
#define NO    0
/*---------------------------------------------------------------------------
 simple filter
 ---------------------------------------------------------------------------*/
static unsigned char value_beyond_limits(int new_val, int old_val, int limit)
{
  if(new_val < (old_val - limit)) return YES;
  if(new_val > (old_val + limit)) return YES;
  return NO;
}

/*---------------------------------------------------------------------------
 adc_handler
 ---------------------------------------------------------------------------*/
void adc_handler(streaming chanend c_logger, chanend c_adc, port trigger_port)
{
  unsigned data[4];          // Array for storing ADC results
  timer adc_trigger_timer;
  unsigned adc_trigger_time;
  sensor_data_t sensor_data_l;

#ifdef CONFIG_PERIODIC
  timer periodic_timer;
  unsigned periodic_interval;
#endif

  sensor_data.ldr = 0;
  sensor_data.joystick_x = 0;
  sensor_data.joystick_y = 0;
  sensor_data.temperature = 0;
  //::adc_config initialise
  at_adc_config_t adc_config = { {0, 0, 0, 0, 0, 0, 0, 0}, 0, 0, 0};
  //::adc_config

  //::enabling input ADC channels
  adc_config.input_enable[0] = 1; //Input 0 is LDR
  adc_config.input_enable[1] = 1; //Input 1 is thermistor
  adc_config.input_enable[2] = 1; //Input 2 is horizontal axis of the joystick
  adc_config.input_enable[3] = 1; //Input 3 is vertical axis of the joystick
  //::input_channels

  //::config sampling rate and mode
  adc_config.bits_per_sample = ADC_8_BPS;
  adc_config.samples_per_packet = 4; //Allow samples to be sent in one hit
  adc_config.calibration_mode = 0;
  //::adc_sampling

  //::enable_adc
  at_adc_enable(analog_tile, c_adc, trigger_port, adc_config);
  at_adc_trigger_packet(trigger_port, adc_config); //Fire the ADC!
  //::trigger_adc

  // Wait till the app handler is ready
  c_logger :> int _;

  adc_trigger_timer :> adc_trigger_time;         //Set timer for first loop tick
  adc_trigger_time += ADC_TRIGGER_PERIOD;

#ifdef CONFIG_PERIODIC
  periodic_timer :> periodic_interval;
  periodic_interval += LOGGER_TRIGGER_PERIOD;
#endif

  while(1)
  {
    select
    {
#pragma ordered
      case adc_trigger_timer when timerafter(adc_trigger_time) :> adc_trigger_time:
      {
        at_adc_trigger_packet(trigger_port, adc_config);    //Trigger ADC
        adc_trigger_time += ADC_TRIGGER_PERIOD;
        break;
      } // case loop_timer to trigger ADC

      case at_adc_read_packet(c_adc, adc_config, data): //if data ready to be read from ADC
      {
        unsigned char flag_ldr = 0, flag_temp = 0, flag_jx = 0, flag_jy = 0; // clear all flags

        flag_ldr = value_beyond_limits(data[0], sensor_data_l.ldr, 1);
        flag_temp = value_beyond_limits(celsius_temperature(data[1]), sensor_data_l.temperature, 1);
        flag_jx = value_beyond_limits(data[2], sensor_data_l.joystick_x, 1);
        flag_jy = value_beyond_limits(data[3], sensor_data_l.joystick_y, 1);

        sensor_data_l.ldr = data[0];           //ADC0 value
        sensor_data_l.temperature = celsius_temperature(data[1]); //ADC1 value
        sensor_data_l.joystick_x  = data[2];   //ADC2 value
        sensor_data_l.joystick_y  = data[3];   //ADC3 value
        sensor_data = sensor_data_l;

#ifdef CONFIG_CHANGE_STATE // send ADC data to logger, if there is any change in state
        if(flag_ldr || flag_temp || flag_jy || flag_jy)
        {
           c_logger <: sensor_data;
        }
#endif
        break;
      } // case at_adc_read_packet

#ifdef CONFIG_PERIODIC // send ADC data to logger, periodically
      case periodic_timer when timerafter(periodic_interval) :> void:
      {
        c_logger <: sensor_data;
        periodic_timer :> periodic_interval;
        periodic_interval += LOGGER_TRIGGER_PERIOD;
        break;
      } // case loop_timer to trigger logger
#endif
    } // select
  } // while(1)
}
```

E.4 Source code for socket_client.xc

```
/**
 * The copyrights, all other intellectual and industrial
 * property rights are retained by XMOS and/or its licensors.
 * Terms and conditions covering the use of this code can
 * be found in the Xmos End User License Agreement.
 *
 * Copyright XMOS Ltd 2014
 *
 * In the case where this code is a modification of existing code
 * under a separate license, the separate license terms are shown
 * below. The modifications to the code are still covered by the
 * copyright notice above.
 *
 **/

#include "socket_client.h"
#include <xs1.h>
#include <string.h>
#include <print.h>
#include <timer.h>

server_config_t server_cfg;
xtcp_connection_t conn;

/*==========================================================================*/
/**
 *  Send data to the server.
 *
 *  \param c_xtcp   channel XTCP
 *  \param buf      character array containing data
 *  \param len      data length
 *  \return         1 for success, 0 for failure
 **/
static int client_send(chanend c_xtcp, unsigned char buf[], int len)
{
  int finished = 0;
  int success = 1;
  int index = 0, prev = 0;
  int id = conn.id;

  xtcp_init_send(c_xtcp, conn);

  while(!finished)
  {
    slave xtcp_event(c_xtcp, conn);

    switch(conn.event)
    {
      case XTCP_NEW_CONNECTION: xtcp_close(c_xtcp, conn); break;
      case XTCP_REQUEST_DATA:
      case XTCP_SENT_DATA:
      {
        int sendlen = (len - index);
        if (sendlen > conn.mss) sendlen = conn.mss;
        xtcp_sendi(c_xtcp, buf, index, sendlen);
        prev = index;
        index += sendlen;
        if (sendlen == 0)
        {
          finished = 1;
        }
        break;
      }
      case XTCP_RESEND_DATA: xtcp_sendi(c_xtcp, buf, prev, (index-prev)); break;
      case XTCP_RECV_DATA:
      {
        slave
        {
          c_xtcp <: 0;
        } // delay packet receive

        if (prev != len)
        success = 0;
        finished = 1;
        break;
      }
      case XTCP_TIMED_OUT:
      case XTCP_ABORTED:
      case XTCP_CLOSED:
      {
        if (conn.id == id)
        {
          finished = 1;
          success = 0;
        }
        break;
      }
      case XTCP_IFDOWN:
      {
        finished = 1;
        success = 0;
        break;
      }
    }
  }
  return success;
}

/*---------------------------------------------------------------------------
 sock_client_set_server_config
 ---------------------------------------------------------------------------*/
void sock_client_set_server_config(server_config_t server_config)
{
  server_cfg.server_ip[0] = server_config.server_ip[0];
  server_cfg.server_ip[1] = server_config.server_ip[1];
  server_cfg.server_ip[2] = server_config.server_ip[2];
  server_cfg.server_ip[3] = server_config.server_ip[3];
  server_cfg.tcp_in_port = server_config.tcp_in_port;
  server_cfg.tcp_out_port = server_config.tcp_out_port;

  printstr("sock_client_set_server_config\n");
}

/*---------------------------------------------------------------------------
 sock_client_init
 ---------------------------------------------------------------------------*/
void sock_client_init(chanend c_xtcp)
{
  xtcp_ipconfig_t ipconfig;
  conn.event = XTCP_ALREADY_HANDLED;
    do
    {
      slave xtcp_event(c_xtcp, conn);
    } while(conn.event != XTCP_IFUP);
  xtcp_get_ipconfig(c_xtcp, ipconfig);

  printstr("sock_client_init - ");
  printstr("IP Address: ");
  printint(ipconfig.ipaddr[0]);printstr(".");
  printint(ipconfig.ipaddr[1]);printstr(".");
  printint(ipconfig.ipaddr[2]);printstr(".");
  printint(ipconfig.ipaddr[3]);printstr("\n");
}

/*---------------------------------------------------------------------------
 sock_client_connect_to_server
 ---------------------------------------------------------------------------*/
void sock_client_connect_to_server(chanend c_xtcp)
```

```
{
  xtcp_listen(c_xtcp, server_cfg.tcp_in_port, XTCP_PROTOCOL_TCP);
  xtcp_connect(c_xtcp, server_cfg.tcp_out_port, server_cfg.server_ip, XTCP_PROTOCOL_TCP);

  conn.event = XTCP_ALREADY_HANDLED;
    do
    {
      slave xtcp_event(c_xtcp, conn);
    } while(conn.event != XTCP_NEW_CONNECTION);

  printstr("sock_client_connect_to_server\n");

}

/*---------------------------------------------------------------------------
 sock_client_send_data
 ---------------------------------------------------------------------------*/
int sock_client_send_data(chanend c_xtcp, char data[])
{
  return client_send(c_xtcp, data, strlen(data));
}

/*---------------------------------------------------------------------------
 sock_client_request_close
 ---------------------------------------------------------------------------*/
void sock_client_request_close(chanend c_xtcp)
{
  char dummy_data[1];
  client_send(c_xtcp, dummy_data, 0);
  xtcp_close(c_xtcp, conn);

  // Wait till the connection is closed
  conn.event = XTCP_ALREADY_HANDLED;
  do
  {
    slave xtcp_event(c_xtcp, conn);
  } while(conn.event != XTCP_CLOSED);
  // Ack the FIN,ACK from host. Let it close.
  delay_milliseconds(100);
}
```

E.5 Source code for ethernet_board_conf.h

```
#ifndef __ethernet_board_conf_h__
#define __ethernet_board_conf_h__

#define ETHERNET_DEFAULT_PHY_ADDRESS 0

#define SMI_COMBINE_MDC_MDIO 1
#define SMI_MDC_BIT 0
#define SMI_MDIO_BIT 1
#define ETHERNET_DEFAULT_TILE tile[1]
#define PORT_ETH_RXCLK on tile[1]: XS1_PORT_1B
#define PORT_ETH_RXD on tile[1]: XS1_PORT_4A
#define PORT_ETH_TXD on tile[1]: XS1_PORT_4B
#define PORT_ETH_RXDV on tile[1]: XS1_PORT_1C
#define PORT_ETH_TXEN on tile[1]: XS1_PORT_1F
#define PORT_ETH_TXCLK on tile[1]: XS1_PORT_1G
#define PORT_ETH_MDIOC on tile[1]: XS1_PORT_4C
#define PORT_ETH_MDIOFAKE on tile[1]: XS1_PORT_8A
#define PORT_ETH_ERR on tile[1]: XS1_PORT_4D

#ifndef ETHERNET_DEFAULT_CLKBLK_0
#define ETHERNET_DEFAULT_CLKBLK_0 on ETHERNET_DEFAULT_TILE: XS1_CLKBLK_1
#endif
```

```
#ifndef ETHERNET_DEFAULT_CLKBLK_1
#define ETHERNET_DEFAULT_CLKBLK_1 on ETHERNET_DEFAULT_TILE: XS1_CLKBLK_2
#endif

#ifndef PORT_ETH_FAKE
#define PORT_ETH_FAKE on ETHERNET_DEFAULT_TILE: XS1_PORT_8C
#endif

#define ETHERNET_DEFAULT_MII_INIT_lite { \
  ETHERNET_DEFAULT_CLKBLK_0, \
  ETHERNET_DEFAULT_CLKBLK_1, \
\
    PORT_ETH_RXCLK,                              \
    PORT_ETH_ERR,                                \
    PORT_ETH_RXD,                                \
    PORT_ETH_RXDV,                               \
    PORT_ETH_TXCLK,                              \
    PORT_ETH_TXEN,                               \
    PORT_ETH_TXD,                                \
    PORT_ETH_FAKE \
}

#define ETHERNET_DEFAULT_SMI_INIT {ETHERNET_DEFAULT_PHY_ADDRESS, \
                                   PORT_ETH_MDIOC}

#endif // __ethernet_board_conf_h__
```

E.6 Source code for server.py

```
import sys
import signal, os
import time
import threading

# Exit the program if Python version used is lower than 2.7.3
if sys.version_info < (2,7,3):
  print('Required Python version 2.7.3 or newer. Exiting!')
  exit(1)

if sys.version_info < (3,0,1):
  # Python version 2.x
  import SocketServer as socketserver
else:
  # Python version 3.x
  import socketserver

# Check for valid IP address
def valid_ip(address):
  try:
    host_bytes = address.split('.')
    valid = [int(b) for b in host_bytes]
    valid = [b for b in valid if b >= 0 and b <= 255]
    return len(host_bytes) == 4 and len(valid) == 4
  except:
    return False

# Get the IP address to run the server on.
# This should be same as HOST computer's static IP address
try:
  g_HOST = sys.argv[1]
  if not valid_ip(g_HOST):
    exit(1)
except:
   print('Please enter a valid Web server IP address. Exiting!')
   exit(1)
```

```
# Global variables
g_interrupted = False
g_program_running = False
g_log_file = 'datalogger.log'

# Keyboard interrupt handler
def kb_handler(signum, frame):
    global g_interrupted
    g_interrupted = True

signal.signal(signal.SIGINT, kb_handler)

# ---------------------------------------------------------------------------
# The TCP handler - receive data from the device and print it on the console
# ---------------------------------------------------------------------------
class xmos_tcp_handler(socketserver.BaseRequestHandler):

  def handle(self):

    global g_temperature
    global g_program_running

    log = open(g_log_file, 'w')
    log.write('+-------------------+-------+-------------+-------------+-------------+\n')
    log.write('|    Date & Time    |  LDR  | Temperature | Joystick(x) | Joystick(y) |\n')
    log.write('+-------------------+-------+-------------+-------------+-------------+\n')

    while True:
      data = self.request.recv(1024).decode()
      g_start_counter = False
      g_program_running = True
      if data:
        for line in data.split('\n'):
          if line:
            print('XMOS:%s' % line)
            if 'LDR' in line:
              log.write(time.strftime('%F %T') + ' |  ' + line[7:10] + '  |     ' + line[26:29] + '     |
                ↪ ' + line[44:47] + '     |     ' + line[53:56] + '     |' + '\n')
              log.flush()
      else:
        print('-----------------------------------------')
        print('Server: Client closed connection ')
        self.request.close()
        break

      if g_interrupted:
        log.close()
        break

# ---------------------------------------------------------------------------
# start_server - wait until the link is up and then start listening
# ---------------------------------------------------------------------------
def start_server():
  global g_interrupted

  PORT = 80

  print('Server: Logging sensor data to %s' % g_log_file)
  print('Server: Waiting to start socket server')
  print('Server: Press CTRL+C to exit.')

  while True:
    socketserver.TCPServer.allow_reuse_address = True
    try:
      server = socketserver.TCPServer((g_HOST, PORT), xmos_tcp_handler)
      print('Server: socket server started with IP address = %s' % g_HOST)
      print('-----------------------------------------')
      server.serve_forever()

    except KeyboardInterrupt:
      g_interrupted = True
      server.socket.close()
      print('Server: Exiting')
      break
    except Exception as e:
      if 'Permission denied' in str(e):
        print('Server: Permssion denied - please run as administrator')
        g_interrupted = True
        break

      # Wait and try again
      time.sleep(1)

# -----------------------------------------------------------------------------
# MAIN
# -----------------------------------------------------------------------------
if __name__ == "__main__":

  t_server = threading.Thread(target=start_server)
  t_server.setDaemon(True)
  t_server.start()

  while True:
    if g_interrupted:
      print('Server: Terminating...')
      break
    pass
```

E.7 Source code for graph.py

```
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import time
import sys
import signal

# Exit the program if Python version used is lower than 2.7.3
if sys.version_info < (2,7,3):
  print('Required Python version 2.7.3 or newer. Exiting!')
  exit(1)

# Global variables
g_interrupted = False

# Keyboard interrupt handler
def kb_handler(signum, frame):
    global g_interrupted
    g_interrupted = True
    exit(1)

signal.signal(signal.SIGINT, kb_handler)

plt.close('all')
fig = plt.figure(figsize=(10, 10), dpi=90, facecolor='w')
plt.suptitle('XMOS Datalogger', fontsize=13, fontweight='bold')

# Create subplots and set graph axis labels
# set_xlabels - will get clear when calling clear() function
# So using text() here
#LDR
ldr_graph = fig.add_subplot(3,1,1)
fig.text(0.45, 0.66, 'Time in Sec')
fig.text(0.035, 0.83, 'LDR',rotation='vertical')
#Temperature
temp_graph = fig.add_subplot(3,1,2)
fig.text(0.45, 0.34, 'Time in Sec')
fig.text(0.035, 0.58, 'Temperature in Deg C',rotation='vertical')
#Joystick
joystick_graph = fig.add_subplot(3,1,3)
fig.text(0.43, 0.02, 'Joystick X-axis')
fig.text(0.035, 0.24, 'Joystick Y-axis',rotation='vertical')

def animate(i):
   pullData = open('datalogger.log','r')
   dataArray = pullData.readlines()[3:]
   ldr_array = []
   temp_array = []
   joystickX_array = []
   joystickY_array = []
   time_in_sec = 0.0
   time_array = []

   for eachLine in dataArray:
      eachLine = eachLine.strip()
      dummy_0,l,t,x,y,dummy_1 = eachLine.split('|')
      ldr_array.append(l)
      temp_array.append(t)
      joystickX_array.append(x)
      joystickY_array.append(y)
      time_array.append(time_in_sec)
      time_in_sec += 0.2

   ldr_graph.clear()  # Clear memory used for plotting LDR graph
   temp_graph.clear()  # Clear memory used for plotting Temperature graph
   joystick_graph.clear()  # Clear memory used for plotting Joystick graph

   # Start plotting Fresh graph
   ldr_graph.plot(time_array, ldr_array, 'r-')   # red dash lines
   temp_graph.plot(time_array, temp_array, 'g-') # green dash lines
   joystick_graph.plot(joystickX_array,joystickY_array,'b-') # blue dash lines

   plt.tight_layout(pad=4, w_pad=4, h_pad=4.5)
ani = animation.FuncAnimation(fig,animate, interval=100)
plt.show()
```