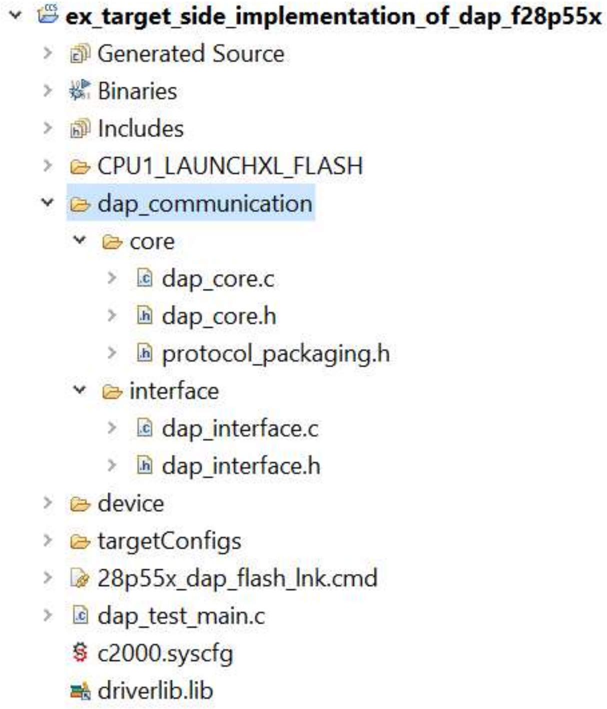
User guide for target_host_comm_protocol library example projects v1

This library contains source code for target side implementation of device agent protocol(dap) to be used to communicate with web-based host GUI.

The source files are categorized into two broder categories:-

1. Core implementation fuctions --> To take care of all backend operation. User need not call these functions directly
2. User interface functions --> User are supposed to call APIs from these files and make relevant changes for their project



```
ex_target_side_implementation_of_dap_f28p55x
    Generated Source
    Binaries
    Includes
    CPU1_LAUNCHXL_FLASH
    dap_communication
        core
            dap_core.c
            dap_core.h
            protocol_packaging.h
        interface
            dap_interface.c
            dap_interface.h
    device
    targetConfigs
    28p55x_dap_flash_lnk.cmd
    dap_test_main.c
    c2000.syscfg
    driverlib.lib
```

Example project list

The following example projects created for user reference to guide them how to use the library files:

1. `ex_target_side_implementation_of_dap_f28p55x`

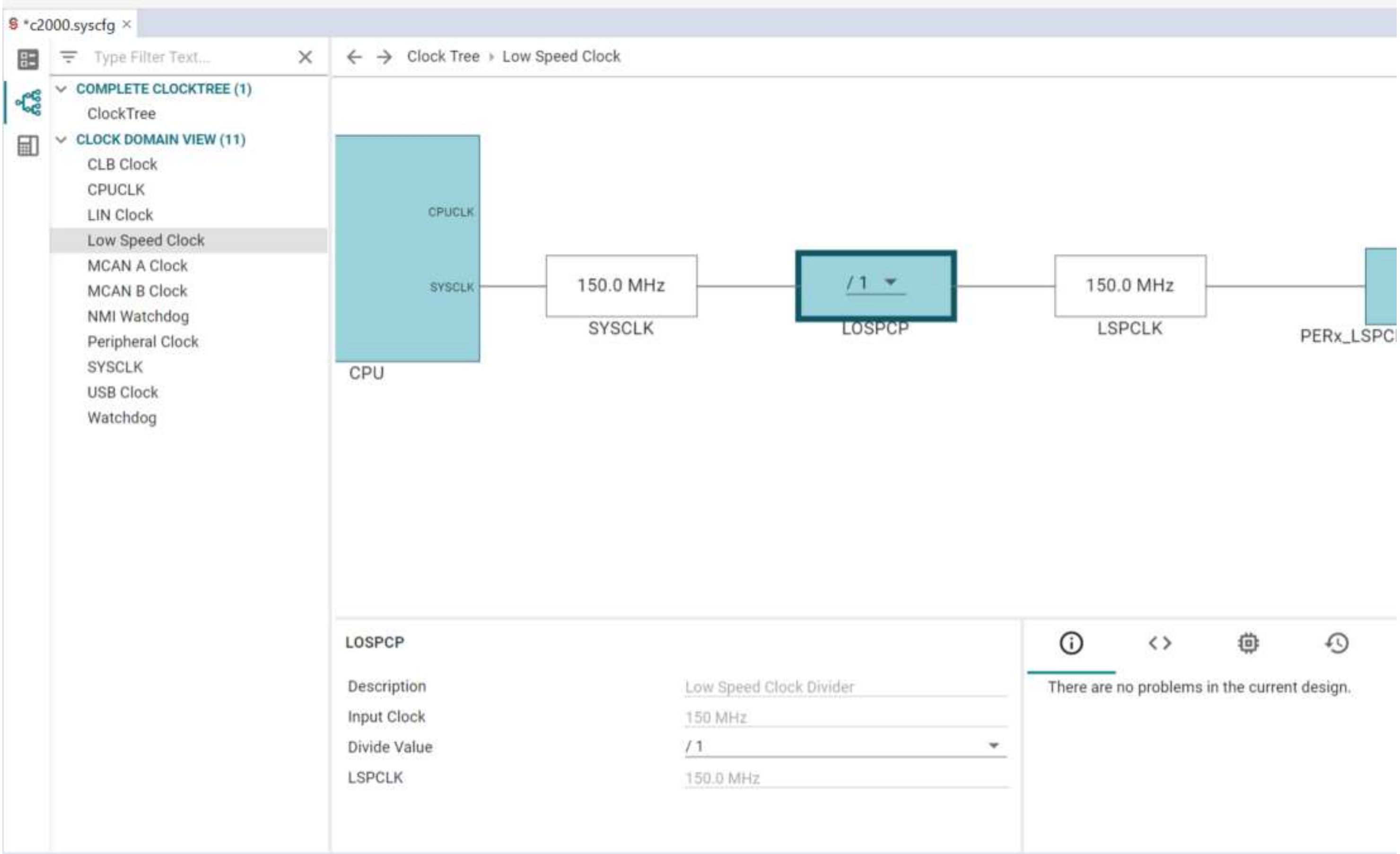
How to run the example project

`Step 1: Set baud rate & property value datatype macro`

In the `dap_interface.h` file set the following macro for necessary baud rate and datatype of property value:-

```
#define SCI_BAUD_RATE 2343750         // 37.5Mhz/16
//#define SCI_BAUD_RATE 9375000         // 37.5Mhz/16 *4
//#define SCI_BAUD_RATE 4687500         // 37.5Mhz/16 *2

#define PROPERTY_VAL_TYPE_UINT16    1   //default value is 1
#define PROPERTY_VAL_TYPE_UINT32    0   //default value is 0
```

Note: Maximum possible baud rate with default `LSPCLK` configuration (`LSPCLK=SYSCLK/4`) is `2343750`. If user wants to run the code for higher baud rate like 4687500 or 9375000, they need to change LSPCLK value from clocktree configuration as shown below:-



LOSPCP

Description Low Speed Clock Divider

Input Clock 150 MHz

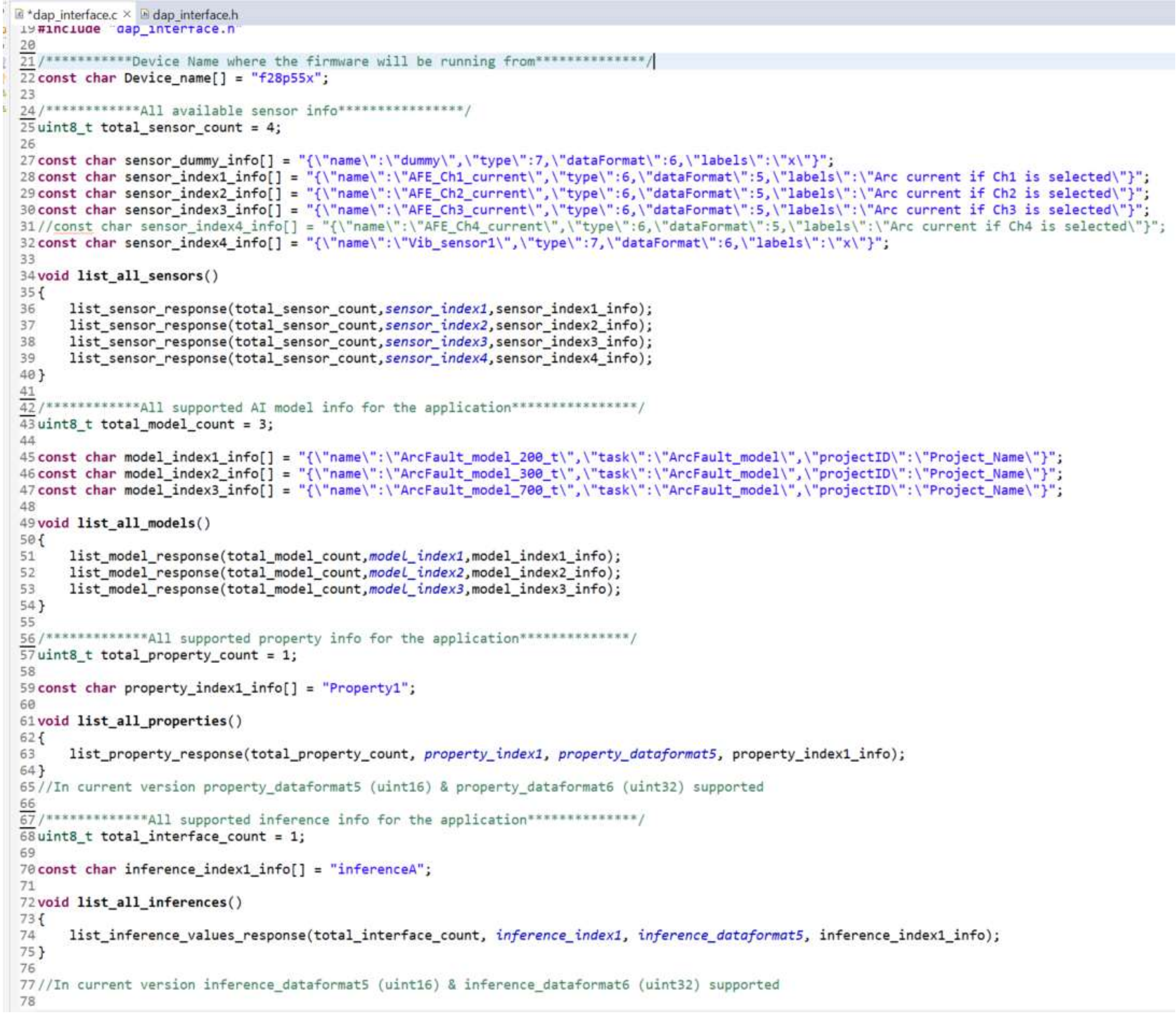
Divide Value / 1

LSPCLK 150.0 MHz

There are no problems in the current design.

`Step 2: sensor, model, property, inference info and device name update`

In the `dap_interface.c` update the relevant information support by the application environment as shown below:-

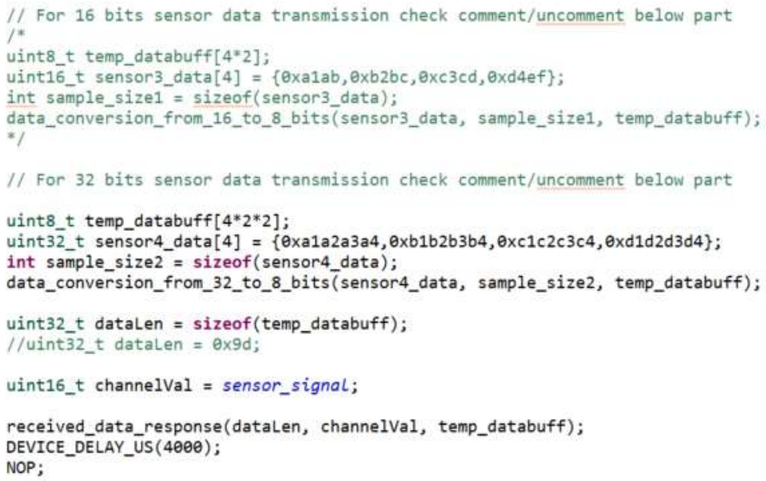


```c
#include "dap_interface.h"

/***********Device Name where the firmware will be running from**************/
const char Device_name[] = "f28p55x";

/************All available sensor info****************/
uint8_t total_sensor_count = 4;

const char sensor_dummy_info[] = "{\"name\":\"dummy\",\"type\":7,\"dataFormat\":6,\"labels\":\"x\"}";
const char sensor_index1_info[] = "{\"name\":\"AFE_Ch1_current\",\"type\":6,\"dataFormat\":5,\"labels\":\"Arc current if Ch1 is selected\"}";
const char sensor_index2_info[] = "{\"name\":\"AFE_Ch2_current\",\"type\":6,\"dataFormat\":5,\"labels\":\"Arc current if Ch2 is selected\"}";
const char sensor_index3_info[] = "{\"name\":\"AFE_Ch3_current\",\"type\":6,\"dataFormat\":5,\"labels\":\"Arc current if Ch3 is selected\"}";
//const char sensor_index4_info[] = "{\"name\":\"AFE_Ch4_current\",\"type\":6,\"dataFormat\":5,\"labels\":\"Arc current if Ch4 is selected\"}";
const char sensor_index4_info[] = "{\"name\":\"Vib_sensor1\",\"type\":7,\"dataFormat\":6,\"labels\":\"x\"}";

void list_all_sensors()
{
    list_sensor_response(total_sensor_count,sensor_index1,sensor_index1_info);
    list_sensor_response(total_sensor_count,sensor_index2,sensor_index2_info);
    list_sensor_response(total_sensor_count,sensor_index3,sensor_index3_info);
    list_sensor_response(total_sensor_count,sensor_index4,sensor_index4_info);
}

/************All supported AI model info for the application***************/
uint8_t total_model_count = 3;

const char model_index1_info[] = "{\"name\":\"ArcFault_model_200_t\",\"task\":\"ArcFault_model\",\"projectID\":\"Project_Name\"}";
const char model_index2_info[] = "{\"name\":\"ArcFault_model_300_t\",\"task\":\"ArcFault_model\",\"projectID\":\"Project_Name\"}";
const char model_index3_info[] = "{\"name\":\"ArcFault_model_700_t\",\"task\":\"ArcFault_model\",\"projectID\":\"Project_Name\"}";

void list_all_models()
{
    list_model_response(total_model_count,model_index1,model_index1_info);
    list_model_response(total_model_count,model_index2,model_index2_info);
    list_model_response(total_model_count,model_index3,model_index3_info);
}

/*************All supported property info for the application**************/
uint8_t total_property_count = 1;

const char property_index1_info[] = "Property1";

void list_all_properties()
{
    list_property_response(total_property_count, property_index1, property_dataformat5, property_index1_info);
}
//In current version property_dataformat5 (uint16) & property_dataformat6 (uint32) supported

/*************All supported inference info for the application**************/
uint8_t total_interface_count = 1;

const char inference_index1_info[] = "inferenceA";

void list_all_inferences()
{
    list_inference_values_response(total_interface_count, inference_index1, inference_dataformat5, inference_index1_info);
}

//In current version inference_dataformat5 (uint16) & inference_dataformat6 (uint32) supported

```

Note: Property value of `uint16_t` and `uint32_t` only supported in this version to support read_property/write_property command response

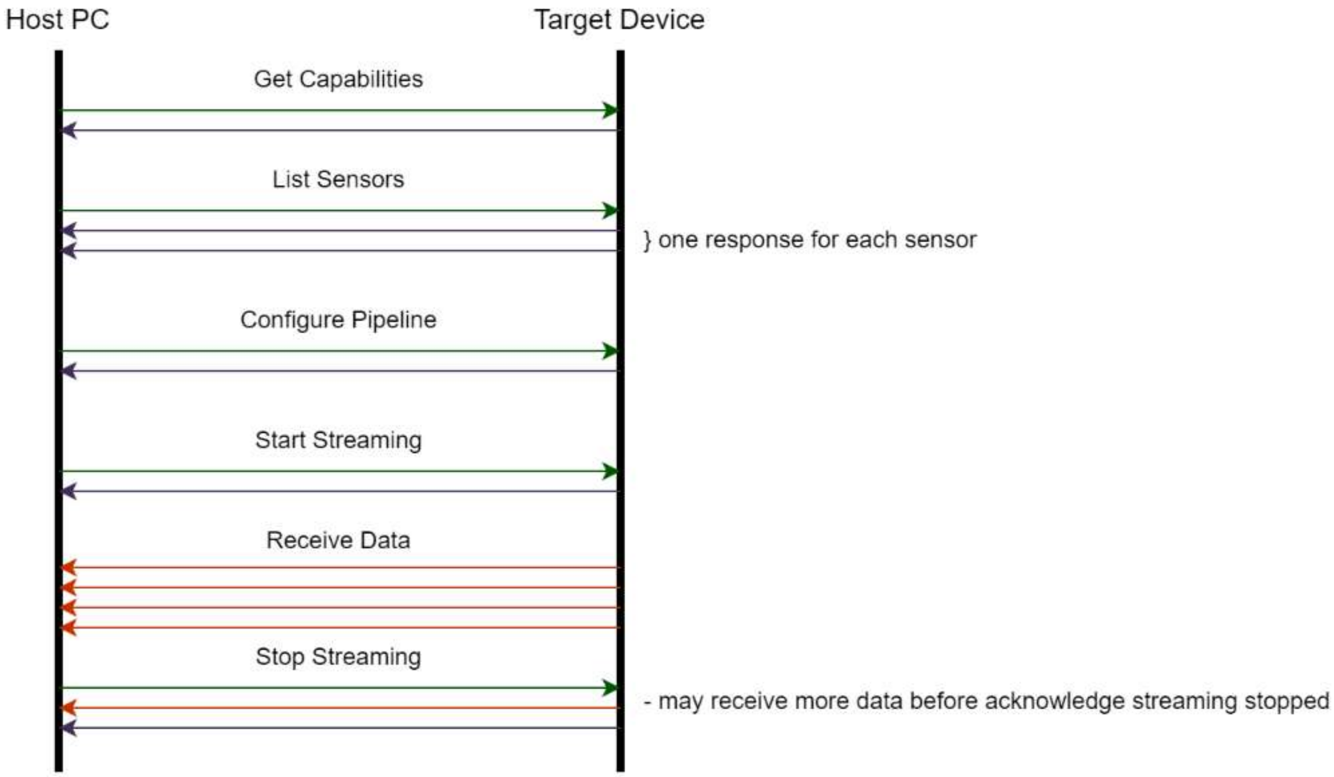
Step 3: sensor, model, property, inference info and device name update

Construct the test data array in `dap_test_main.c` file and call the `data_conversion_from_16_to_8_bits(uint16_t* input_buf, int input_buf_size, uint8_t* output_buf)`/`data_conversion_from_32_to_8_bits(uint32_t* input_buf, int input_buf_size, uint8_t* output_buf)` as per the test datatype to convert them in `uint8_t` format. Finally call the `received_data_response(uint32_t data_payload_length, uint16_t channel_value, uint16_t* data_array)` API to execute data transmission:



```
// For 16 bits sensor data transmission check comment/uncomment below part
/*
uint8_t temp_databuff[4*2];
uint16_t sensor3_data[4] = {0xa1ab,0xb2bc,0xc3cd,0xd4ef};
int sample_size1 = sizeof(sensor3_data);
data_conversion_from_16_to_8_bits(sensor3_data, sample_size1, temp_databuff);
*/

// For 32 bits sensor data transmission check comment/uncomment below part

uint8_t temp_databuff[4*2*2];
uint32_t sensor4_data[4] = {0xa1a2a3a4,0xb1b2b3b4,0xc1c2c3c4,0xd1d2d3d4};
int sample_size2 = sizeof(sensor4_data);
data_conversion_from_32_to_8_bits(sensor4_data, sample_size2, temp_databuff);

uint32_t dataLen = sizeof(temp_databuff);
//uint32_t dataLen = 0x9d;

uint16_t channelVal = sensor_signal;

received_data_response(dataLen, channelVal, temp_databuff);
DEVICE_DELAY_US(4000);
NOP;
```

Example usecase of packet tramission between host & target will be similar as shown below where host will always initiate the command & target will respond back to that accordingly.



Reference

1. Details regarding the protocal is documented in: https://confluence.itg.ti.com/display/EDGEST/Serial+Communication+Protocol#SerialCommunicationProtocol-DataFormat.1