rcl\_allocator\_t allocator = rcl\_get\_default\_allocator();

// create init\_options

rclc\_support\_t support;

RCCHECK(rclc\_support\_init(&support, 0, NULL, &allocator));

// create node

rcl\_node\_t node;

RCCHECK(rclc\_node\_init\_default(&node, "add\_twoints\_client\_rclc", "", &support));

Or

RCCHECK(rclc\_node\_init\_default(&node, "zephyr\_int32\_publisher", "", &support));

Or

RCCHECK(rclc\_node\_init\_default(&node, "zephyr\_tof\_node", "", &support));

// create executor

rclc\_executor\_t executor;

RCCHECK(rclc\_executor\_init(&executor, &support.context, 1, &allocator));

//rclc Node finish

RCCHECK(rcl\_node\_fini(&node));

**Add One Publisher**

rcl\_publisher\_t publisher;

std\_msgs\_\_msg\_\_Int32 msg;

void timer\_callback(rcl\_timer\_t \* timer, int64\_t last\_call\_time)

{

RCLC\_UNUSED(last\_call\_time);

if (timer != NULL) {

RCSOFTCHECK(rcl\_publish(&publisher, &msg, NULL));

msg.data++;

}

}

Main()

Rcl allocator

Rclc support init

Create node

// create publisher

RCCHECK(rclc\_publisher\_init\_default(&publisher,&node,ROSIDL\_GET\_MSG\_TYPE\_SUPPORT(std\_msgs, msg, Int32),"zephyr\_int32\_publisher"));

Or

RCCHECK(rclc\_publisher\_init\_default(&tof\_publisher, &node, ROSIDL\_GET\_MSG\_TYPE\_SUPPORT(std\_msgs, msg, Float32), "/sensors/tof"));

// create timer,

rcl\_timer\_t timer;

const unsigned int timer\_timeout = 1000;

RCCHECK(rclc\_timer\_init\_default(&timer,&support,RCL\_MS\_TO\_NS(timer\_timeout),timer\_callback));

Or

RCCHECK(rclc\_timer\_init\_default(&timer, &support, RCL\_MS\_TO\_NS(10), timer\_callback));

Create Executor

//If Publisher add, executor+timer

RCCHECK(rclc\_executor\_add\_timer(&executor, &timer));

For publish several time, use while loop

msg.data = 0;

while(1){

rclc\_executor\_spin\_some(&executor, RCL\_MS\_TO\_NS(100));

usleep(100000);

}

RCCHECK(rcl\_publisher\_fini(&publisher, &node))

Rcl node finish

**Int32 wifi publisher**

// Wireless management

static struct net\_mgmt\_event\_callback wifi\_shell\_mgmt\_cb;

static void wifi\_mgmt\_event\_handler(struct net\_mgmt\_event\_callback \*cb,uint32\_t mgmt\_event, struct net\_if \*iface){

if(NET\_EVENT\_IPV4\_ADDR\_ADD == mgmt\_event){

printf("DHCP Connected\n");

connected = 1;

}

}

//For Publisher

Timer callback function(){}

Main()

// ------ Wifi Configuration ------

net\_mgmt\_init\_event\_callback(&wifi\_shell\_mgmt\_cb,wifi\_mgmt\_event\_handler,NET\_EVENT\_IPV4\_ADDR\_ADD);

net\_mgmt\_add\_event\_callback(&wifi\_shell\_mgmt\_cb);

struct net\_if \*iface = net\_if\_get\_default();

static struct wifi\_connect\_req\_params cnx\_params;

cnx\_params.ssid = "WIFI\_SSID\_HERE";

cnx\_params.ssid\_length = strlen(cnx\_params.ssid);

cnx\_params.channel = 0;

cnx\_params.psk = "WIFI\_PSK\_HERE";

cnx\_params.psk\_length = strlen(cnx\_params.psk);

cnx\_params.security = WIFI\_SECURITY\_TYPE\_PSK;

if (net\_mgmt(NET\_REQUEST\_WIFI\_CONNECT, iface, &cnx\_params, sizeof(struct wifi\_connect\_req\_params)))

{

printf("Connection request failed\n");

} else {

printf("Connection requested\n");

}

//for network connection while loop

while (!connected)

{

printf("Waiting for connection\n");

usleep(10000);

}

printf("Connection OK\n");

// ------ micro-ROS ------

Rcl allocator

Rclc support init

Create node

Create Publisher

Create Timer

Create Executor

Execute spin some

Publisher finish

Node finish

**Open Manipulator TOF**

One publisher = sensor

rcl\_publisher\_t tof\_publisher;

void timer\_callback(rcl\_timer\_t \* timer, int64\_t last\_call\_time)

{

RCLC\_UNUSED(last\_call\_time);

if (timer != NULL) {

struct sensor\_value value;

std\_msgs\_\_msg\_\_Float32 msg;

//Sensor fetch and get(dev value + msg data(value.val1+value.val2)

sensor\_sample\_fetch(dev);

sensor\_channel\_get(dev, SENSOR\_CHAN\_DISTANCE, &value);

msg.data= ((float)value.val1 + (float)value.val2)/1000.0;

rcl\_publish(&tof\_publisher, (const void\*)&msg, NULL);

}

}

Main()

//LED and Sensor Configuration

led = device\_get\_binding(DT\_GPIO\_LABEL(DT\_ALIAS(led0), gpios));

gpio\_pin\_configure(led, PIN, GPIO\_OUTPUT\_ACTIVE | 0);

dev = device\_get\_binding(DT\_LABEL(DT\_INST(0, st\_vl53l1x)));

if (dev == NULL) {

printf("Could not get VL53L1X device\n");

}

Rcl allocator

Rclc support init

Create node

Create TOF publisher

Create timer

Create executor

For publish several time, use while loop

**3 Publisher + 2 subscriber**

//publisher

rcl\_publisher\_t tof\_publisher;

rcl\_publisher\_t imu\_publisher;

rcl\_publisher\_t trigger\_publisher;

//subscriber

rcl\_subscription\_t led\_subscription;

rcl\_subscription\_t thr\_subscription;

Timer callback for publisher

void timer\_callback(rcl\_timer\_t \* timer, int64\_t last\_call\_time)

{

RCLC\_UNUSED(last\_call\_time);

if (timer != NULL) {

//two sensor publish

RCSOFTCHECK(rcl\_publish(&tof\_publisher, &tof\_data, NULL));

RCSOFTCHECK(rcl\_publish(&imu\_publisher, &imu\_data, NULL));

trigger\_msg.data = tof\_data.data\*1000 < threshold;

RCSOFTCHECK(rcl\_publish(&trigger\_publisher, &trigger\_msg, NULL));

}

}

//Subscription callback(msg->data)

void led\_subscription\_callback(const void \* msgin)

{

const std\_msgs\_\_msg\_\_Bool \* msg = (const std\_msgs\_\_msg\_\_Bool \*)msgin;

gpio\_pin\_set(led, PIN, (int)(msg->data) ? 1 : 0);

}

void thr\_subscription\_callback(const void \* msgin)

{

const std\_msgs\_\_msg\_\_Int32 \* msg = (const std\_msgs\_\_msg\_\_Int32 \*)msgin;

threshold = msg->data;

}

Main()

//LED and sensors binding

// ---- Sensor configuration ----

led = device\_get\_binding(DT\_GPIO\_LABEL(DT\_ALIAS(led0), gpios));

gpio\_pin\_configure(led, PIN, GPIO\_OUTPUT\_ACTIVE | 0);

struct device \*tof\_sensor = device\_get\_binding(DT\_LABEL(DT\_INST(0, st\_vl53l0x)));

struct sensor\_value tof\_value;

struct device \*imu\_sensor = device\_get\_binding(DT\_LABEL(DT\_INST(0, st\_lsm6dsl)));

struct sensor\_value imu\_value;

struct sensor\_value accel\_x, accel\_y, accel\_z;

// set accel/gyro sampling frequency to 104 Hz

imu\_value.val1 = 104;

imu\_value.val2 = 0;

//IMU Sensor Attribute(imu value, acceleration channel, gyro channel)

**Add two integer service**

Two service = request and response

Request = a + b

Response = sum

void service\_callback(const void \* req, void \* res){

example\_interfaces\_\_srv\_\_AddTwoInts\_Request \* req\_in = (example\_interfaces\_\_srv\_\_AddTwoInts\_Request \*) req;

example\_interfaces\_\_srv\_\_AddTwoInts\_Response \* res\_in = (example\_interfaces\_\_srv\_\_AddTwoInts\_Response \*) res;

printf("Service request value: %d + %d.\n", (int) req\_in->a, (int) req\_in->b);

res\_in->sum = req\_in->a + req\_in->b;

}

Main()

Rcl allocator

Rclc support init

Create node

// create service

rcl\_service\_t service;

RCCHECK(rclc\_service\_init\_default(&service, &node, ROSIDL\_GET\_SRV\_TYPE\_SUPPORT(example\_interfaces, srv, AddTwoInts), "/addtwoints"));

Create executor

Executor add service

example\_interfaces\_\_srv\_\_AddTwoInts\_Response res;

example\_interfaces\_\_srv\_\_AddTwoInts\_Request req;

RCCHECK(rclc\_executor\_add\_service(&executor, &service, &req, &res, service\_callback));

rclc\_executor\_spin(&executor);

RCCHECK(rcl\_service\_fini(&service, &node));

Rcl node finish

-------xxxxx-------xxxxxx-----------xxxxx--------

Firmware/zephyrproject/zephyr/subsys/net/ip/net\_if.c

struct net\_if \*net\_if\_get\_default(void)

{

struct net\_if \*iface = NULL;

if (\_net\_if\_list\_start == \_net\_if\_list\_end) {

return NULL;

}

#if defined(CONFIG\_NET\_DEFAULT\_IF\_ETHERNET)

iface = net\_if\_get\_first\_by\_type(&NET\_L2\_GET\_NAME(ETHERNET));

#endif

#if defined(CONFIG\_NET\_DEFAULT\_IF\_IEEE802154)

iface = net\_if\_get\_first\_by\_type(&NET\_L2\_GET\_NAME(IEEE802154));

#endif

#if defined(CONFIG\_NET\_DEFAULT\_IF\_BLUETOOTH)

iface = net\_if\_get\_first\_by\_type(&NET\_L2\_GET\_NAME(BLUETOOTH));

#endif

#if defined(CONFIG\_NET\_DEFAULT\_IF\_DUMMY)

iface = net\_if\_get\_first\_by\_type(&NET\_L2\_GET\_NAME(DUMMY));

#endif

#if defined(CONFIG\_NET\_DEFAULT\_IF\_OFFLOAD)

iface = net\_if\_get\_first\_by\_type(NULL);

#endif

#if defined(CONFIG\_NET\_DEFAULT\_IF\_CANBUS\_RAW)

iface = net\_if\_get\_first\_by\_type(&NET\_L2\_GET\_NAME(CANBUS\_RAW));

#endif

#if defined(CONFIG\_NET\_DEFAULT\_IF\_CANBUS)

iface = net\_if\_get\_first\_by\_type(&NET\_L2\_GET\_NAME(CANBUS));

#endif

#if defined(CONFIG\_NET\_DEFAULT\_IF\_PPP)

iface = net\_if\_get\_first\_by\_type(&NET\_L2\_GET\_NAME(PPP));

#endif

return iface ? iface : \_net\_if\_list\_start;

}

Firmware/zephyrproject/zephyr/include/net/wifi\_mgmt.h

struct wifi\_connect\_req\_params

{

uint8\_t \*ssid;

uint8\_t ssid\_length; /\* Max 32 \*/

uint8\_t channel;

uint8\_t \*psk;

uint8\_t psk\_length; /\* Min 8 - Max 64 \*/

enum wifi\_security\_type security;

};