ROEL

ON/OFF : 416ED707

{4750,4700, 550,1500, 600,1500, 600,1500, 550,1500, 600,1500, 550,1550, 550,2600, 600,1500, 550,1500, 600,1500, 550,2600, 600,1500, 600,1500, 550,1500, 600,2600, 550,2600, 600,1500, 4750,4650, 600,1500, 550,1500, 550,1550, 600,1500, 550,1500, 600,1500, 600,2550, 600,1500, 550,1550, 550,1500, 600,2600, 550,1500, 600,1500, 600,1500, 550,2600, 600,2600, 550}; // UNKNOWN 416ED707

MIST : 83821F1C

{4800,4650, 550,1500, 550,1550, 550,1550, 550,1500, 550,1550, 550,1550, 500,2650, 550,1550, 500,1550, 550,2650, 550,1500, 550,2650, 550,1500, 550,1550, 550,2650, 500,1550, 550,1550, 4750,4650, 550,1550, 550,1500, 550,1550, 550,1550, 550,1500, 550,1550, 550,2600, 550,1550, 550,1550, 550,2600, 550,1550, 550,2600, 550,1550, 550,1550, 500,2650, 550,1550, 500}; // UNKNOWN 83821F1C

HUMIDITY : D0955537

{4750,4650, 550,1550, 550,1550, 500,1550, 550,1550, 550,1550, 500,1550, 550,2650, 500,1550, 550,2650, 550,1500, 550,1550, 550,2600, 550,1550, 550,1550, 500,1550, 550,2650, 550,1500, 4800,4650, 550,1550, 500,1550, 550,1550, 550,1500, 550,1550, 550,1550, 500,2650, 550,1550, 550,2600, 550,1500, 600,1500, 550,2650, 550,1550, 500,1550, 550,1550, 550,2600, 550}; // UNKNOWN D0955537

ANION : EAADC1DC

{4750,4700, 500,1550, 550,1550, 600,1500, 500,1550, 500,1600, 500,1550, 550,2650, 600,1500, 500,2650, 500,1600, 500,2650, 500,1650, 450,2650, 550,1550, 500,1600, 500,1550, 600,1500, 4700,4700, 500,1600, 500,1550, 550,1550, 500,1600, 500,1550, 550,1550, 500,2650, 550,1550, 500,2700, 500,1550, 500,2700, 500,1550, 550,2650, 500,1550, 550,1550, 500,1600, 500}; // UNKNOWN EAADC1DC

TIMER : ACD56894

{4750,4700, 550,1500, 550,1550, 550,1500, 600,1500, 600,1500, 550,1550, 500,2650, 550,1550, 500,1550, 600,1500, 550,1550, 500,2650, 550,2650, 500,2650, 550,1550, 500,1550, 550,1550, 4750,4650, 550,1550, 550,1550, 500,1550, 550,1550, 550,1500, 550,1550, 550,2650, 500,1550, 550,1550, 550,1500, 550,1550, 550,2650, 500,2650, 550,2650, 500,1550, 550,1550, 550}; // UNKNOWN ACD56894

SLEEP MODE : 7E5BBE2C unsigned int rawData[71] = {4750,4650, 550,1550, 500,1600, 500,1550, 550,1550, 500,1600, 500,1550, 550,2650, 500,1550, 550,1550, 550,2600, 550,2650, 550,1500, 550,1550, 550,2650, 500,1550, 550,1550, 550,1500, 4750,4700, 550,1500, 550,1550, 550,1550, 500,1550, 550,1550, 550,1550, 500,2650, 550,1550, 500,1550, 550,2650, 550,2600, 550,1550, 550,1550, 500,2650, 550,1550, 500,1550, 550}; // UNKNOWN 7E5BBE2C

Carrier 에어컨

온도 설정(▼) : B24DBF40

온도 설정(▲) : B24DBF40 (위 아래 같음)

운전선택 :

자동-B24D1FE0냉방-B24DBF40제습-B24D1FE0난방-B24DBF40송풍-B24DBF40

운전/정지 : B24DBF40 / B24D7B84

바람세기 :

약-B24D9F60

중-B24D5FA0

강-B24D3FC0

풍향조절(상) : B24D0FF0

풍향조절(하) : B24D6B94

취침운전(on/off) : B24DE01F/B24DBF40

타이머(상) : B24D9F60

타이머(하) : B24DBF40

My Mode : B24D1FE0

Follow Me : BA4559A6

LED : B54AF50A

Turbo : B54AF50A (led와 같음)

공기청정기

전원 48B7C837

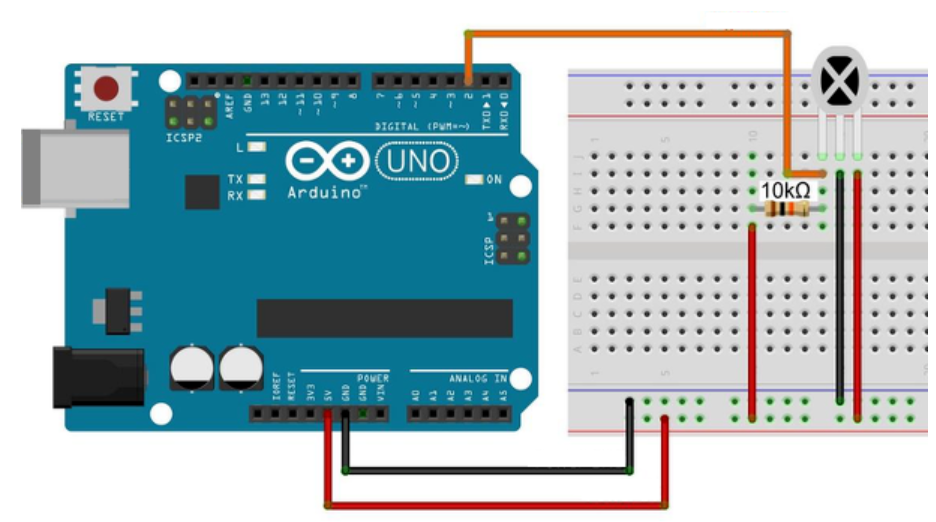
Turbo 48B748B7

Auto 48B76897

Fan Speed

-Up 48B78877

-Down 48B708F7



//------------------------------------------------------------------------------

// Include the IRremote library header

//

#include <IRremote.h>

//------------------------------------------------------------------------------

// Tell IRremote which Arduino pin is connected to the IR Receiver (TSOP4838)

//

int recvPin = 11;

IRrecv irrecv(recvPin);

//+=============================================================================

// Configure the Arduino

//

void setup ( )

{

Serial.begin(9600); // Status message will be sent to PC at 9600 baud

irrecv.enableIRIn(); // Start the receiver

}

//+=============================================================================

// Display IR code

//

void ircode (decode\_results \*results)

{

// Panasonic has an Address

if (results->decode\_type == PANASONIC) {

Serial.print(results->address, HEX);

Serial.print(":");

}

// Print Code

Serial.print(results->value, HEX);

}

//+=============================================================================

// Display encoding type

//

void encoding (decode\_results \*results)

{

switch (results->decode\_type) {

default:

case UNKNOWN: Serial.print("UNKNOWN"); break ;

case NEC: Serial.print("NEC"); break ;

case SONY: Serial.print("SONY"); break ;

case RC5: Serial.print("RC5"); break ;

case RC6: Serial.print("RC6"); break ;

case DISH: Serial.print("DISH"); break ;

case SHARP: Serial.print("SHARP"); break ;

case JVC: Serial.print("JVC"); break ;

case SANYO: Serial.print("SANYO"); break ;

case MITSUBISHI: Serial.print("MITSUBISHI"); break ;

case SAMSUNG: Serial.print("SAMSUNG"); break ;

case LG: Serial.print("LG"); break ;

case WHYNTER: Serial.print("WHYNTER"); break ;

case AIWA\_RC\_T501: Serial.print("AIWA\_RC\_T501"); break ;

case PANASONIC: Serial.print("PANASONIC"); break ;

case DENON: Serial.print("Denon"); break ;

}

}

//+=============================================================================

// Dump out the decode\_results structure.

//

void dumpInfo (decode\_results \*results)

{

// Check if the buffer overflowed

if (results->overflow) {

Serial.println("IR code too long. Edit IRremoteInt.h and increase RAWBUF");

return;

}

// Show Encoding standard

Serial.print("Encoding : ");

encoding(results);

Serial.println("");

// Show Code & length

Serial.print("Code : ");

ircode(results);

Serial.print(" (");

Serial.print(results->bits, DEC);

Serial.println(" bits)");

}

//+=============================================================================

// Dump out the decode\_results structure.

//

void dumpRaw (decode\_results \*results)

{

// Print Raw data

Serial.print("Timing[");

Serial.print(results->rawlen-1, DEC);

Serial.println("]: ");

for (int i = 1; i < results->rawlen; i++) {

unsigned long x = results->rawbuf[i] \* USECPERTICK;

if (!(i & 1)) { // even

Serial.print("-");

if (x < 1000) Serial.print(" ") ;

if (x < 100) Serial.print(" ") ;

Serial.print(x, DEC);

} else { // odd

Serial.print(" ");

Serial.print("+");

if (x < 1000) Serial.print(" ") ;

if (x < 100) Serial.print(" ") ;

Serial.print(x, DEC);

if (i < results->rawlen-1) Serial.print(", "); //',' not needed for last one

}

if (!(i % 8)) Serial.println("");

}

Serial.println(""); // Newline

}

//+=============================================================================

// Dump out the decode\_results structure.

//

void dumpCode (decode\_results \*results)

{

// Start declaration

Serial.print("unsigned int "); // variable type

Serial.print("rawData["); // array name

Serial.print(results->rawlen - 1, DEC); // array size

Serial.print("] = {"); // Start declaration

// Dump data

for (int i = 1; i < results->rawlen; i++) {

Serial.print(results->rawbuf[i] \* USECPERTICK, DEC);

if ( i < results->rawlen-1 ) Serial.print(","); // ',' not needed on last one

if (!(i & 1)) Serial.print(" ");

}

// End declaration

Serial.print("};"); //

// Comment

Serial.print(" // ");

encoding(results);

Serial.print(" ");

ircode(results);

// Newline

Serial.println("");

// Now dump "known" codes

if (results->decode\_type != UNKNOWN) {

// Some protocols have an address

if (results->decode\_type == PANASONIC) {

Serial.print("unsigned int addr = 0x");

Serial.print(results->address, HEX);

Serial.println(";");

}

// All protocols have data

Serial.print("unsigned int data = 0x");

Serial.print(results->value, HEX);

Serial.println(";");

}

}

//+=============================================================================

// The repeating section of the code

//

void loop ( )

{

decode\_results results; // Somewhere to store the results

if (irrecv.decode(&results)) { // Grab an IR code

dumpInfo(&results); // Output the results

dumpRaw(&results); // Output the results in RAW format

dumpCode(&results); // Output the results as source code

Serial.println(""); // Blank line between entries

irrecv.resume(); // Prepare for the next value

}

}