Prototype of the Linux based embedded DC Motor driver system regulated through analog input signal with positional angle reading using magnetometer compass module.

Tomasz Chądzyński
Department of Electrical Engineering
San Jose State University
Email: tomasz.chadzynski@sjsu.edu

Abstract—**Embedded devices gain increasing popularity thanks to the widespread of IoT and Cyber-Physical systems. Ever growing computing power and miniaturization allows for the construction of systems that offer increased performance, sophistication and can be used in a growing number of small single-purpose devices. The presented project focuses on utilizing the ARM Embedded computing platform Smart4418 for a signal processing using the ADC, I2C bus communication and control of stepper motor with PWM. The project uses a modern multi-tasking Linux operating system to provide high reliability and concurrent operating environment as a base for custom device drivers and control software. The presented project uses a series of circuits tasked with receiving user input through the variable voltage control system and transfers it into the control signal for a stepper motor dictating its speed and direction. On the software side, the project utilizes signal compensation techniques to provide a stable and reliable interpretation of an analog signal. Additionally, the quantized signal is subjected through analysis using the Discrete Fourier Transform and power spectrum. The signal analysis proper sampling and minimal noise interference. As other function, the projects utilize a magnetometer device to calculate current heading of the device and an industry grade NEMA17 stepper motor controller through the device PWM functionality. The work presents the use of modern hardware-software tools and techniques in constructing high performance cyber-physical embedded systems.**

Index Terms—**Embedded Hardware, Linux, Kernel Drivers, ARM, GPIO, PWM, Hardware Design, C/C++, Python, FFT, Cyber-Physical systems, LSM303, I2C**

1. Introduction

The projects contain two design goals. The first goal is to construct a simple mechanism based on embedded computing platform capable of processing analog input and use it to direct the motion of a DC stepper motor. The second goal is to adapt the LSM303 magnetometer device and use it as a compass to detect the current heading of the device. The project is based on an ARM SOC from Samsung the S5P4418 ARM microprocessor. The SOC contains many peripherals allowing for interaction with the external environment. Among the resources available are the GPIO capabilities for establishing digital communication with external devices, Analog to Digital converter capable of processing the input signal in the form of electric potential difference, ranging from 0 to 1.8 Volt, also series pins equipped with PWM function for generating uniform shape square wave and built-in I2C bus master allowing for serial communication with external devices.

The SOC platform is capable of executing the Linux operating system used as the base for the software-side control system. The Linux is a multitasking operating system implementing the POSIX API programming interface. The system executes many tasks simultaneously thus allowing for the construction of control systems designed to service multiple inputs and outputs devices simultaneously. The Kernel of the system provides a large variety of supporting device drivers and comes with the board support package for the S5P4418 SOC.

The approach taken to implement the project is emphasizing user-space system services controlling high-level functions with the support from low-level kernel device drivers. The user-space application is deployed as a UNIX "daemon"1 which connects to the device drivers through various interfaces like character device or IIO [2]. The daemon application runs continuously, processing the input data and excreting control signals to drive the output devices.

2. Hardware Configuration

The hardware setup used in the project is a combination of various modules with the central control unit being the ARM S5P4418 based system called Smart4418 with a companion board the FriendlyARM 1606. The S5P4418 is an ARM Cortex A9 CPU capable of dynamic frequency scaling from 400MHz to 1.4 GHz. The system offers 1GB DDR3 RAM and a number of input/output peripherals.

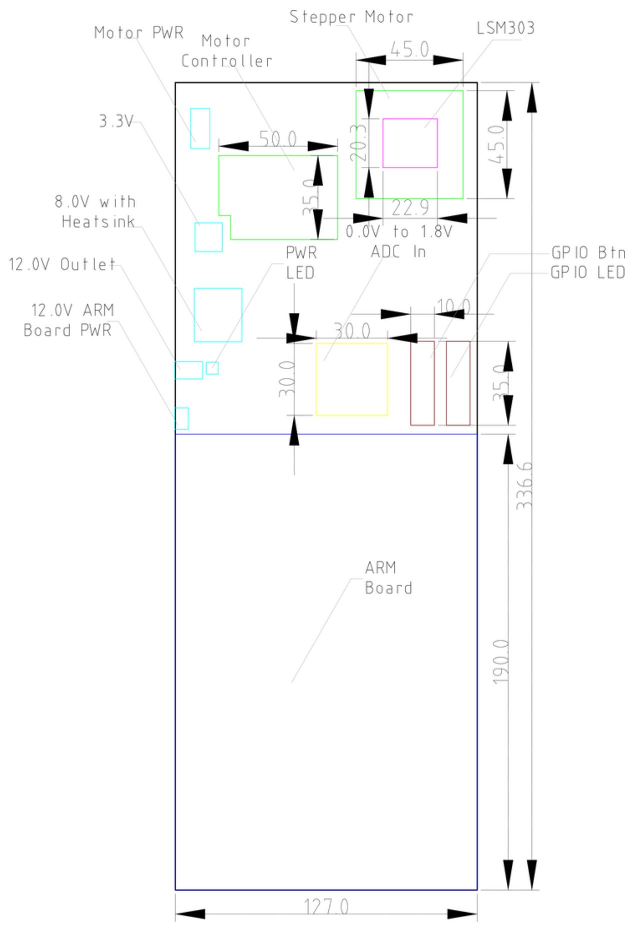

Figure 1. Board Floor Plan

2.1. Board Layout

The board layout is shown in Figure.1 presents the general overview of the system components placement. All the elements are mounted on a through hole prototyping board which serves as a mounting and connecting platform. The bottom part of the board is occupied exclusively by the Smart4418SDK. The top part houses all the user input and output components. The left side of the top portion is dedicated to the power distribution section supplying the SDK and components. The DC stepper motor and the DC Stepper motor driver both occupy the top part of the board while the GPIO button and LED along with the ADC input are situated in the bottom part close to the SDK board. The LSM303 magnetometer is mounted on the stepper motor. A 10cm long shaft is used to provide sufficient distance between the magnetometer and the motor such that the magnetic field of the motor would not interfere with magnetometer readings.

2.2. Power Distribution

The power distribution section focuses on power delivery to all the components in the system. The section contains four major components. Two linear regulators supplying

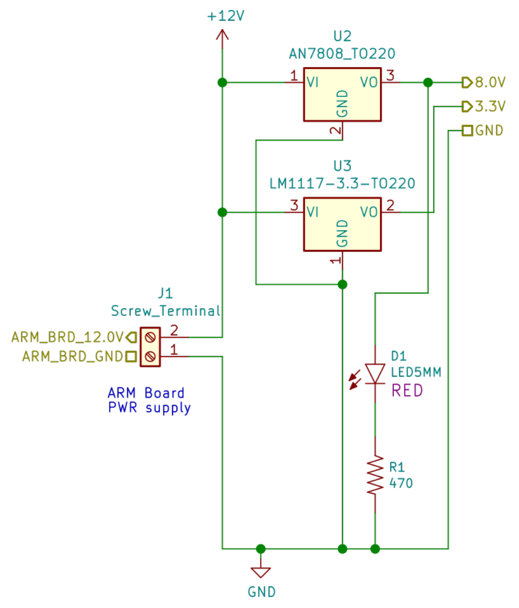

Figure 2. Power Distribution

3.3V and 8.0V and an LED indicating the device is connected to the power supply. The required power supply for the system is the 12V DC regulated supply. The LED uses approximately $13mA$ of current as a result obtained from the Formula.1 derived through the KCL loop method.

$$I_{LED} = \frac{V_D - V_{LED}}{R1} = \frac{8 - 1.8}{470} = .0131A \approx 13mA \quad (1)$$

Aside from providing the 8V and 3.3V supply the section also redirects the 12V supply directly to the ARM board. A double-contact screw terminal allows for attaching the 5mm plug and connects the ARM SDK to a single power supply.

The AN7808 8V linear regulator purpose is mainly to supply power to the DC motor driver. As such the maximum expected current to the driver is at the limit of 750mA. To avoid overheating the regulator is additionally supplied with a heat sink to provide better heat dissipation characteristics during high current operation.

2.3. GPIO Input Button and Output LED

The GPIO interfacing circuit shown in Figure.6 consists of two parts. The input part is a push-button connecting the GPIO input to the 3.3v through a 220Ω resistor. The 220Ω resistor serves as current limiting circuit protection in the case of misconfiguration of the GPIO pin and accidental short to the ground.

The second circuit element is the output LED diode. The diode is connected directly to the 3.3V source and then to the ground through 470Ω resistor and an N-MOSFET transistor. The use of the transistor minimizes current that needs to be soured from the ARM SDK board as the SDK pins are designed to provide signal but not power to the components.

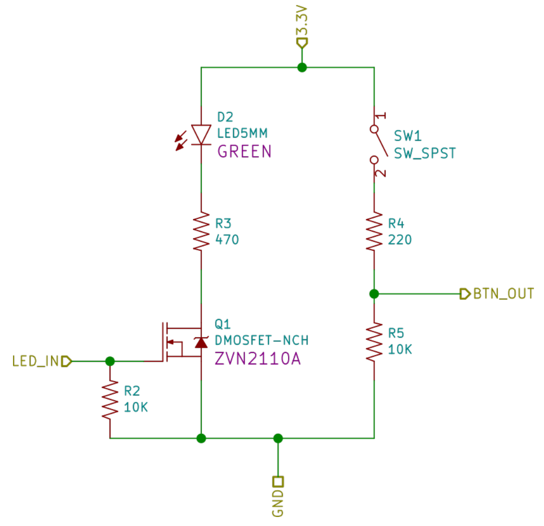

Figure 3. GPIO interfacing, input button and output LED

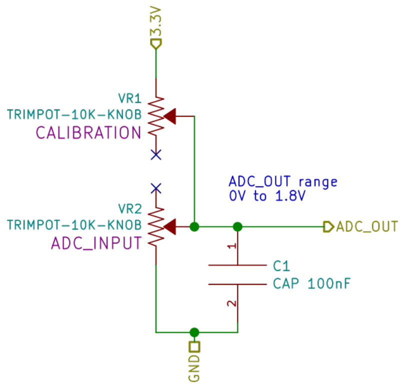

Figure 4. ADC input circuit

2.4. ADC Input

The ADC input circuit shown in Figure.4 consist of two variable resistors and a smoothing capacitor. The top resistor VR1 serves as a calibration knob that allows the stepping down of the maximum voltage from 3.3V to 1.8V. The lower resistor serves as a tuning node for the input signal fed into the ADC. The circuit operates in the voltage range from 0V to 1.8V that matches the dynamic range of the ADC.

2.5. DC Stepper Motor Driver

The DC motor driver circuit shown in Figure.5 is based on the EasyDriver v4.5 module. The circuit provides a simple interface to control a variety of DC motors. The unit operates as the chopper driver module with the maximum current capability of 750mA. The motor driver module connects to the AN7808 8V linear voltage regulator through an external switch that allows for safe disconnecting the subsystem while maintaining power delivery to other components. The unit uses four inputs the MS1 and MS2 allow for configuration of the step size of the motor and in current

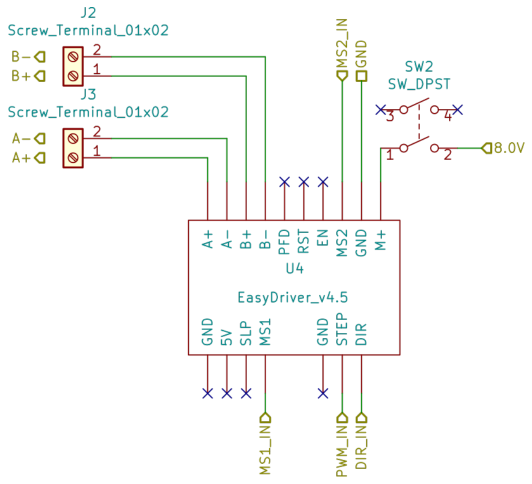

Figure 5. Stepper motor driver

configuration are both connected to the ground setting the largest step. The driving inputs STEP and DIR connect to the PWM output of the SDK board and the GPIO pin dedicated to specifying the rotation direction. The unit provides four connections A+, A-, B+ and B- to attach a bipolar stepper motor. The connectors provide modular mounting using the screw terminals.

2.6. LSM303 Magnetometer

The LSM303 magnetometer used in the project serves as a compass module and detects the current heading of the device X-axis [3]. The module communicates with the main CPU using the I2C bus. The project uses an application of LSM303 provided by Adafruit Inc. The application consists of the LSM303 chipset with pre-assembled supporting components on a small reusable PCB board. The component provides pinout necessary to connect the device. This project communicates with the magnetometer module through two wires, a bidirectional SDA Data line, and single direction clock SCL. Also, the module connects to the 3.3V power line and common ground. The connection is shown in Figure.6.

2.7. Global System Connectivity

Figure.6 shows a global view of connections between each system module. The power unit distributes 3.3V to the GPIO, and ADC modules then 8.0V to the DC motor driver and finally, pass through 12.0V to the SDK board. The GPIO module uses the SDK GPIO ports GPIO31 and GPIO29. The ADC module connects directly to the ADC3 input of the Smart4418, and the stepper motor driver utilizes the GPIO13 for direction and PWM0 for rotation speed. The LSM303 magnetometer connects to the SDA0 and SCL0 pins of the main ARM SDK board.

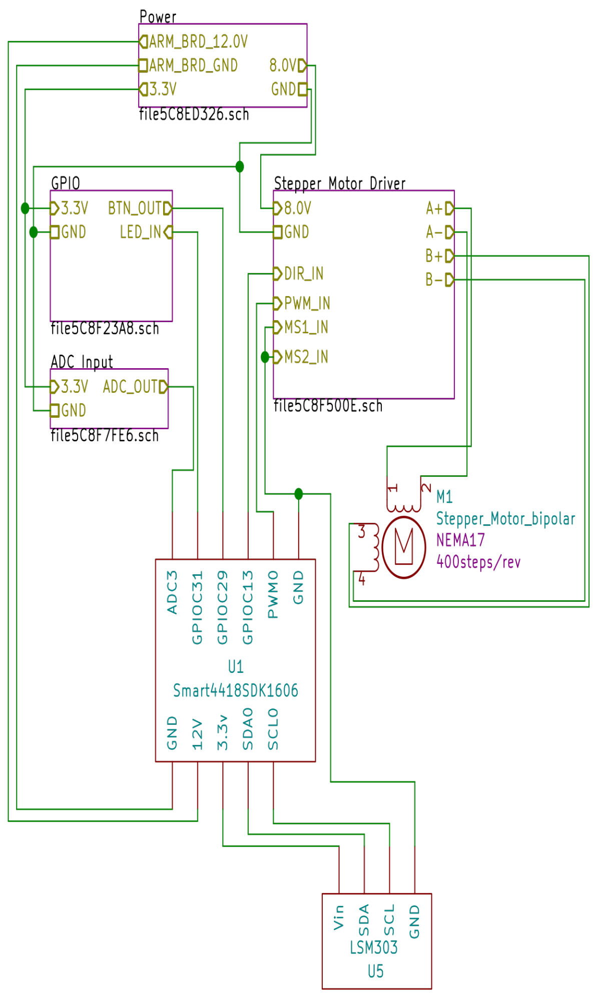

Figure 6. System connectivity

3. Building the Software for the Target System.

Building the working software for the ARM system requires the use of the cross compiler provided by FriendlyARM. The project code base, as well as the busy box package, are built using the tools provided by the board supplier. The full listing from the build process of project one is shown in Appendix B. To build the software for ARM platform it is necessary to cross-compile the source code. The project uses a Linux standard build system called autoconf which provides configurable, automated setup of the building process. The Listing.1 shows the configuration step that uses the cross compiler and sets up the build process for the target platform ARM on the host x86 machine.

Listing 1. Configuration step from the cross compilation of the project 2

```
../../ee242_sw/prj2/configure --build=x86_64
    -pc-linux-gnu --host=arm-cortexa9-linux-
    gnueabihf CROSS_COMPILE=arm-linux- --
    prefix=/usr
```

4. GPIO Interfacing

The board support package for the Smart4418 platform provides drivers for the GPIO input/output system. The drivers are available as part of the kernel in the form of GPIO controller drivers [2]. The GPIO controller drivers are part of the "gpio_chip" interface which is not directly available from the user-space. Additional character driver was required to provide an interface to the GPIO pins. The used Linux Kernel version 4.4 allowed to use GPIO legacy framework [2] to establish the connection to the GPIO pin controllers and expose the GPIO functionality through custom driver "tomasz_gpio.c". The project 2 leverages the same GPIO framework to provide the direction signal for the stepper motor driver as well as the activation function for motor movement and LED indicating the system is working.

5. ADC Interfacing and Compensation

The Analog to Digital interfacing uses the built-in ADC converter provided with the ARM SOC. The maximum sampling rate of the ADC is one million samples per second with 12-bit resolution and dynamic range from 0V to 1.8V [4]. The ADC input is connected to external signal source shown in Figure.4. The potentiometer VR2 provides the DC voltage signal to the ADC within the dynamic range through adjustment of the position. Then, the voltage is processed by the ARM SDK.

The client application uses provided device driver included with the system board support package. The driver exposes the user-space interface based on the IIO bus standard. The client application reads the latest sampled voltage from the "in_voltage3_raw" file exposed as part of the "sys" filesystem. The raw output range in the ADC readout ranges from 0 corresponding to 0V at the input to 4095 which equals 1.8V.

5.1. ADC Input Compensation

Processing input voltage using built-in ADC provides a relatively simple way to digitize analog signals but carries inherent inaccuracies of the measurements. The discrepancies come from minimal non-linear properties of the electronic devices and the noise. Figure.7 shows the relationship of experimentally obtained data points compared to the ideal linear characteristic of the dynamic range. The minimal differences show the high quality of the ADC unit; however, to fully correct the probed signal it is necessary to apply the signal compensation.

The compensation of a raw signal is based on the calculation of a second linear function that corrects the reading within the given range. The base concept is shown by Equation.2. A raw sample represented by the function $f(x)$ is added to the compensation function $g(x)$ effectively resulting in a linear characteristic with 0 offset and ideal slope a defined by Equation.3.

$$y(x) = f(x) + g(x) = (px + q) + (a_g x + b_g) = ax \quad (2)$$

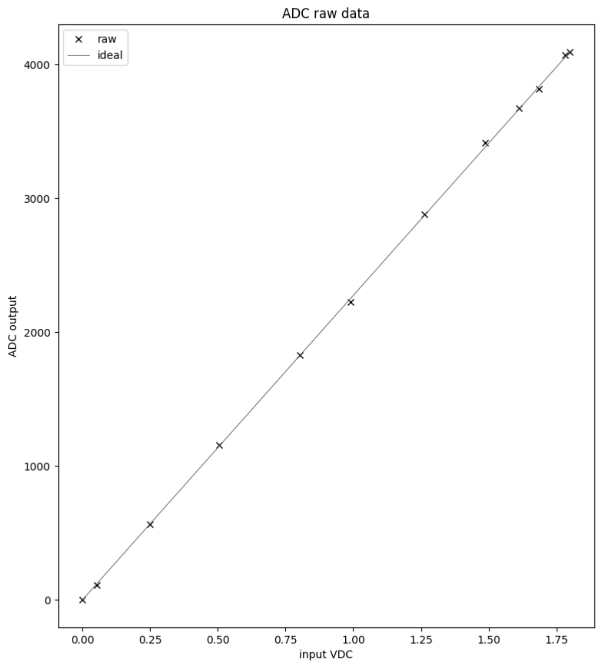

Figure 7. ADC Raw data vs Ideal

The used compensation scheme assumes that the target ideal linear characteristic has the y-intercept component equal zero meaning the value of ideal function $y(x)$ is equal to zero at the origin.

$$a = \frac{y\_max_{ideal} - y\_min_{ideal}}{x\_max_{ideal} - x\_min_{ideal}} \tag{3}$$

The algorithm operates on a set of input pairs composed from the measured input voltage and the ADC readout. The result is a piece-wise defined function $g(x)$. Processing occurs using two neighbouring pairs of values. The first step computes the piece-wise slope and bias of the raw function $f(x)$ show in Equations 4 and 5

$$p = \frac{y_{2\_raw} - y_{1\_raw}}{x_{2\_raw} - x_{1\_raw}} \tag{4}$$

$$q = y_{1\_raw} - p * x_{1\_raw} \tag{5}$$

The obtained pair of p and q is valid only for the range between the two input pairs. A separate slope and bias needs to be calculated for each segment across the dynamic range. Next step is to obtain the characteristic of the compensating $g(x)$ function. The formulas 6 and 7 are used to obtain the slope and bias of the correction used to compensate raw input data. ‘

$$a_g = a - p \tag{6}$$

$$b_g = -q \tag{7}$$

Figure.8 shows the effect of compensation applied to the initial point. Corrected data points coincide exactly the ideal slope of the dynamic range proving the correctness of the algorithm.

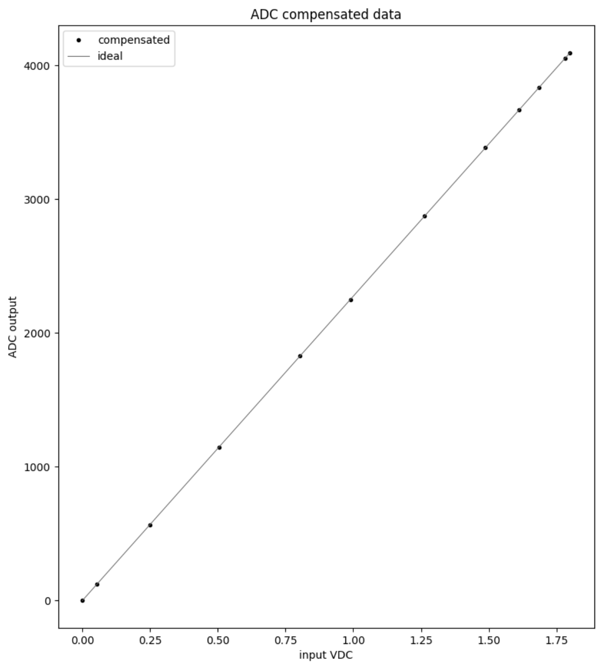

Figure 8. ADC Compensated data vs Ideal

The result of the algorithm is an array of piece-wise pairs of slope and bias values of the correcting function $g(x)$. The calculated data are supported with corresponding ADC raw output ranges used to identify which segment should be applied to correct currently processed value. The calculation of $g(x)$ function happens outside of the main working service program. The data is pre-computed using a python script. Part of the script that pre-computes the function is shown in Listing.2.

The pre-computed data is then saved in a data file and transferred to the ARM SDK. When the service program starts, it's compensation module reads the array and uses for real-time correction. In the service program the data for the $g(x)$ function is stored within an array of structures shown on Listing.3.

The *comp_model* structure is the main data object that keeps the array of *comp_point* structures containing the data. When an ADC reading arrives, it is present in the form of $f(x)$. The initial step searches the data array to find a corresponding *comp_point* structure. The search takes advantage of the fact that all the structures are organized

Listing 2. Calculation of $g(x)$

```
def compute_g(i_d):
    ret = []

    a = (i_d.raw_max - i_d.raw_min)/(i_d.
        input_max - i_d.input_min)

    for i in range(0, len(i_d.raw_points)-1)
        :
        c = i_d.raw_points[i]
        n = i_d.raw_points[i+1]

        p = (n.raw_out-c.raw_out)/(n.v_in-c.
            v_in);
        q = c.raw_out - p*c.v_in
        ag = a-p
        bg = -q

        point = Point()
        point.p = p
        point.q = q
        point.ag = ag
        point.bg = bg
        point.range_begin = c.raw_out
        point.range_end = n.raw_out

        #cheat a little add one point to
            last range to make the search
            easier
        if i == len(i_d.raw_points)-2:
            point.range_end += 1

        ret += [point]

    return ret
```

Listing 3. Model of $g(x)$

```
typedef struct _comp_point {
        float p;
        float q;
        float ag;
        float bg;
        uint32_t range_begin;
        uint32_t range_end;
} comp_point;

typedef struct _comp_model {
        uint32_t n;
        comp_point* lut;
} comp_model;
```

Listing 4. Binary search iterative version

```
uint32_t comp_search_by_raw_out(const
    comp_model *model, adc_raw val)
{
        /*
         * Assuming correct input data
         * Binary search by raw ADC data
         */
        uint32_t begin = 0;
        uint32_t end = (model->n) - 1;

        while( end != begin) {
                uint32_t m = comp_mid(begin,
                     end);
                int pos = comp_check_range
                    (&(model->lut[m]),val);

                if(pos == 0) {
                        return m;
                }
                else if(pos == -1) {
                        end = m-1;
                }
                else { // pos == 1
                        begin = m + 1;
                }
        }

        return begin;
}
```

in ascending order with respect to the dynamic range. Therefore, the binary search is applied instead of a linear search. Use of the binary search allows for reduction of algorithmic complexity from $O(n)$ to $O(log(n))$. Since the data processing happens in real time, it is essential to apply optimizations where possible to reduce the delay of the operation.

The binary search algorithm shown in Listing.4 is an iterative implementation targeted toward embedded platforms. Although the recursive version might provide a more elegant solution in terms of code readability, the overhead resulting from the function call reduces the overall performance, and it is not suitable for embedded real-time applications. The supporting functions $comp_mid$ and $comp_check_range$ provide a method to check if a given raw ADC output is within the range of currently examined segment of the $g(x)$ function and compute midpoint in the array segment used in the binary search. The complete source code of both functions is present in the attached archive. When a corresponding piece-wise element is found, the second step uses the original p and q to obtain the voltage fed to the ADC as shown in Equation.8.

$$x = f^{-1}(y) = \frac{y - q}{p} \tag{8}$$

After obtaining the original input voltage x the $g(x)$ is computed and added with existing $f(x)$ that produces the compensated result as shown in Equation.9.

Listing 5. Computing compensated ADC reading

```
float comp_compensate(const comp_model*
    model, adc_raw f)
{
        //Find the corresponding range
        uint32_t idx_range =
            comp_search_by_raw_out(model, f);
        comp_point r = model->lut[idx_range
            ];

        //Use p and q to get original input
        float v_in = (f-r.q)/r.p;

        //compute g(x)
        float g = v_in*r.ag + r.bg;

        //Rerurn compensated f+g
        return f+g;
}
```

$$y_{final}(x) = f(x) + g(x) \tag{9}$$

The compensated value calculation is realized by one function in the service application code running on the ARM SDK. The source code of the function is shown in Listing.5

Applying compensation allows adjusting the values captured by ADC to reduce the quantization errors. This functionality is especially important in systems where precision is a significant requirement in the specification of the cyber-physical system. Computing the correction function is a relatively computationally efficient task under the condition that the data of function $g(x)$ is previously pre-computed and the size of the lookup table is not too large.

6. Analysis of signal quality using Discrete Fourier Transform and Power Spectrum

Verification of the sampled analog signal is a critical point to obtain insight into the quality of the resulting dataset. Quantization carries a number of problems including noise and aliasing that could potentially take over or alter the original information from the analog source. Further analysis of signal quality is necessary to determine whether how much the original message is affected and if there is a need for additional filtering.

An example of a signal with noise is shown in Figure.9. The signal comes from the sampled ADC input circuit(Figure.4) with the data gathered by one of the modes of operation within the service application running on the ARM SDK.

Usually, examining the obtained signal in the time domain becomes a difficult task. Most of the time the expected shape of the signal is unknown, the duration of the signal could stretch over a period of time that makes it impossible to examine the nature of the information adequately.

Much more practical approach is to convert the signal from time domain into frequency domain and perform the

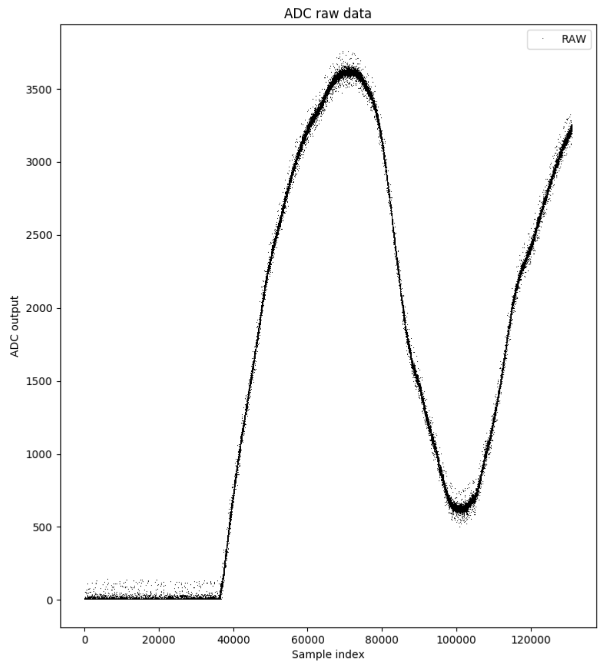

Figure 9. Sampled raw data

analysis on the translated form. The conversion into time domain of a digital signal is performed using the Discrete Fourier Transform [5] or in practical applications, the implementation of DFT called Fast Fourier Transform [5].

The DFT formula is shown in the Equation.10. The entire range of the signal is traversed producing a set of orthogonal with respect to each other components representing specific frequency harmonics of the signal.

$$X[m] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-j2\pi(n*m)/N} \tag{10}$$

The DFT produces results represented by the complex numbers. Each harmonic provides information about the magnitude and phase of the signal at a given frequency. Specifically for the purposes of validating signal contamination by noise and occurrence of aliasing, the complex number representation introduces unnecessary complexity as the phase information is not needed to analyze the strength of the signal alone. To obtain the insight into the signal amplitude with respect to frequency, the computation of the power spectrum is performed on the frequency domain representation obtained by using the FFT transform. A single component in the power spectrum is represented by the magnitude of the complex number obtained using Equation.11.

$$P[m] = \sqrt{Re^2[x[m]] + Im^2[x[m]]} \tag{11}$$

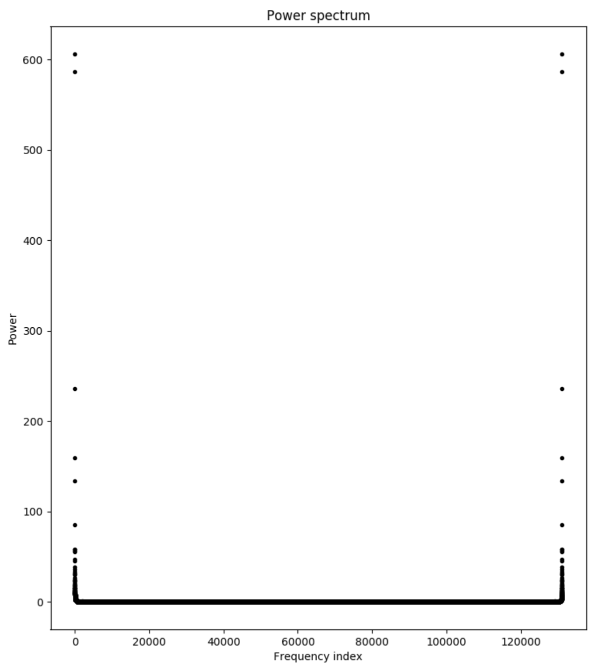

Figure 10. Power spectrum without DC component

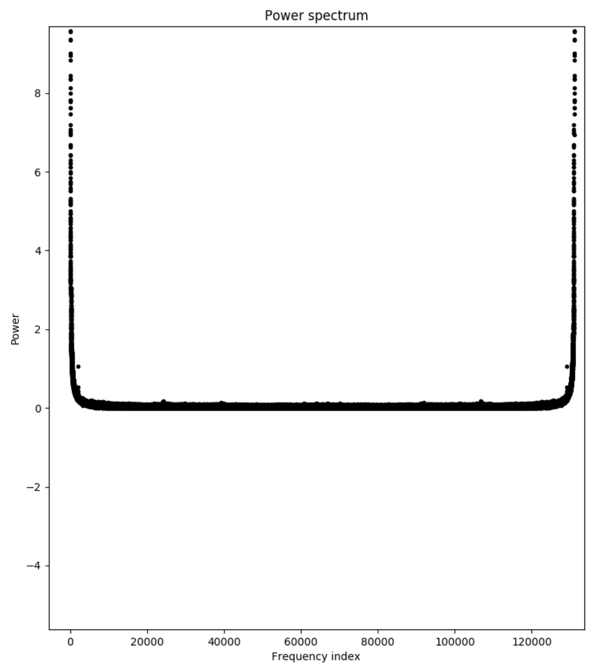

Figure 11. Zoom in on Power spectrum showing low magnitude noise

The calculation is performed for each frequency component resulting in a set of elements representing the power of the signal with respect to the corresponding frequency. The power spectrum for the signal in Figure.9 is shown in the Figure.10.

The data presented in the power spectrum does not contain the DC component present at index 0. A significant low-frequency contribution is present as expected from the low raising signal. The high-frequency range of the spectrum appears to be at the zero level indicating minimal or none high-frequency noise presence. Zooming closer to the zero level shown in Figure.11 reveals minimal high-frequency noise presence. However, the noise magnitude is well below 1.

The power spectrum shows nearly non-existent high-frequency noise presence and no aliasing. The power spectrum confirms that the signal was sampled at sufficient sampling frequency and the quality of connection allows for clear capture of signal with almost non-existent interference. Also, the shape of the power spectrum appears to be reflected across the center point indicating the expected and correct result. For the signal to be correctly sampled the Nyquist criterium shown in Equation.12 must be satisfied.

$$f_{sampling} \geq 2f_{max} \tag{12}$$

The rule states that the sampling frequency must be at least twice as big as the highest frequency component of the sampled signal. For the discussed data, the period between samples is equal to $10\mu s$ with 2^17 total samples. The highest sampled frequency is approximately equal to $100kHz$ which can correctly sample signal with the highest frequency harmonic of $50kHz$. Looking at the signal in Figure.9, the shape resembles a shifted up sine wave.

As the influence of noise on the signal is in practice impossible to avoid, it is necessary to estimate the quality of the quantized signal. One method to estimate how much contribution to the information comes from the information signal and how much the information is distorted by high-frequency noise is to use the energy ratio index shown by Equation.13.

$$\eta = \frac{\Delta_1}{\Delta_{\sum}} \tag{13}$$

The equation is a ratio of the sum of energies from low frequency to the highest expected frequency at which the signal exists to the sum of all existing frequencies within the power spectrum. Formulas 14 and 15 show the way of calculating the sums of energies.

$$E_{total} = \Delta_{\sum} = \sum_{m=0}^{\frac{N}{2}-1} P(m) \tag{14}$$

$$E_{m0} = \Delta_1 = \sum_{m=0}^{m_0} P(m) \ for \ 1 \leq m_0 \leq \frac{N}{2} - 1 \tag{15}$$

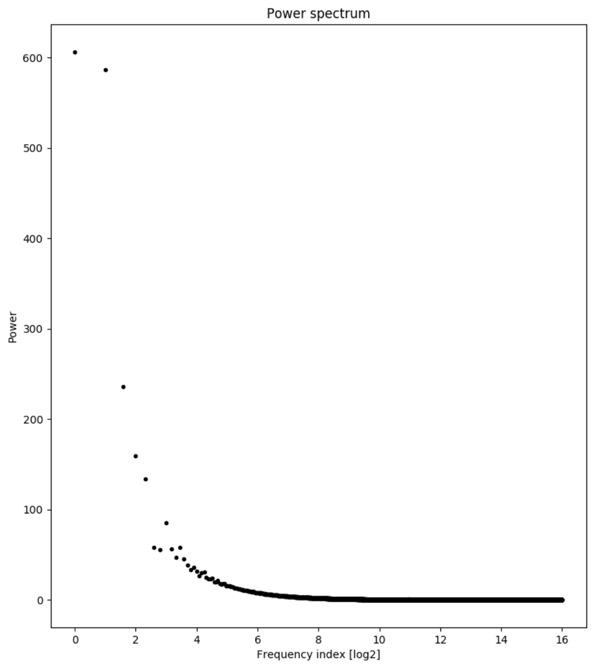

Figure 12. Power spectrum without DC component

During the sampling experiment the System obtained $N = 2^{17}$ samples with time between sample equal to $10\mu s$. The highest represented frequency is $f_{hi} = \frac{1}{10\mu s} \approx 100kHz$. The power of the highest frequency is $P[\frac{N}{2} - 1] = P[65534] = 0$. The power of DC component is $P[0] = 1561$. The expected highest signal frequency is estimated to be present at .2 of the half of the power spectrum from Figure.10 which gives the frequency of $100kHz * 20\% = 20kHz$. The ratio index of energies from DC to $m_0 = 13106$ to total energies from DC to 65535 is shown in Equation.16. The index shows that 80% of information in the sampled spectrum comes from signal where 20% of information comes from high frequency noise.

$$\eta = 0.798896 \approx 80\% \tag{16}$$

The results can be verified by looking at Figure.12. The plot uses a logarithmic scale to distinguish between low frequency and high-frequency components more clearly. The spectrum shows frequencies from the component index 1 (excluding DC component) up to $\frac{N}{2} - 1$. Nearly all of the components with high power value are situated at the left side of the plot. The low-frequency components up to the 2^6th frequency index contain most of the power across the spectrum. Analyzing the results, the signal is correctly sampled with non-existent interference from the noise.

Listing 6. Device tree modification(s5p4418-nanopi2-rev103.dts)

```
&i2c_0 {
        status = "okay";
        mag_meter: lsm303dlhc_mag@1e {
                compatible = "stm,
                    lsm303dlhc_mag";
                reg = <0x1e>;
                status = "okay";
        };
};
```

7. PWM Interfacing

The Linux support for Smart4418 PWM is provided as the part of the board support package from Nexell. The PWM is implemented as a driver using Linux standardized PWM framework [2]. The PWM framework allows for control the channels both through the "sys" file system or from within the kernel space by using the Linux PWM API [2]. This project implementation uses the internal API as the means to communicate with the PWM line. The project provides a character device driver "tomasz_prj1.c" exposing the interface for the service application to control the motor. The driver combines the PWM functionality along with the GPIO to provide the step and dir signals to the motor driver control circuit shown in Figure.5.

8. Compass heading data using LSM303 module

The LSM303DLHC magnetometer is used in the project as a compass showing the heading of the device X-axis with reference to earth magnetic north. The magnetometer device provides measurement data using three magnitudes in the X, the Y and the Z directions. To calculate the compass heading under the assumption that the Z-axis points up, the Formula.17 is used. The formula gives the angle in degrees of the X-axis with respect to the magnetic north.

$$heading = atan2(\frac{Y}{X}) * \frac{180}{2\pi} \tag{17}$$

The LSM303 device is connected to the main ARM board through the I2C bus. As such the device drivers are not present in the original board support package and proper support must be implemented. The taken implementation approach is split between the thin kernel driver and userspace client. The first step is to add the device information into the device tree. The fragment containing modification is shown in Figure.6.

From Figure.13 we see that the LSM303 is connected to pins SDA0 and SCL0 which indicate I2C bus 0. The code fragment references the i2c_0 bus definition and injects LSM303 description into the structure. The reg property provides device address on the bus which is $0x1e$. The device is identified by the driver matching within the kernel through the compatible string. In this case, the LSM303 magnetometer part is known to the kernel as "lsm202dlhc_mag".

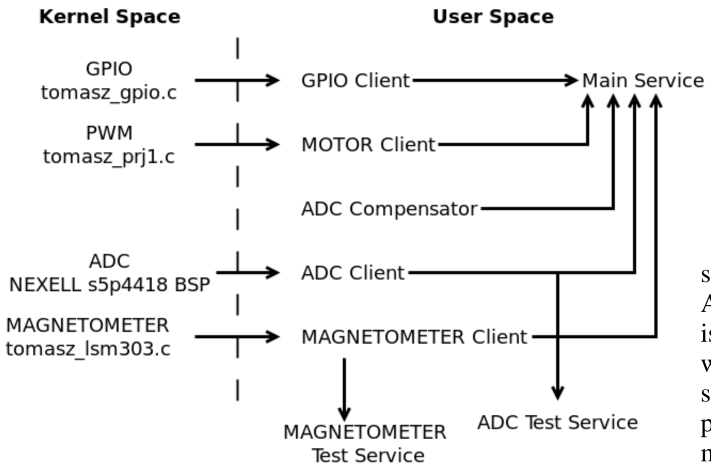

Figure 13. Software structure of the system

The kernel driver provided in reference code inside the file "tomasz_lsm303.c" is a standard driver using the I2C and the character device kernel frameworks. The driver initializes the device and verifies that it is the magnetometer using IRA_REG_M register. Then initializes a polling thread that continuously checks with the device for new data. The thread reads SR_REG_M register and checks the new data ready flag then updates driver data structures with the new measurement.

The character device portion provides the userspace interface. A single read from the device character file provides a copy of new data from the magnetometer. To prevent overloading the system, when the userspace process reads from the driver data is marked as old. Any subsequent reading attempt blocks the function until new data arrives from the device.

9. Overall Program structure

The overall structure of the implementation is organized following the standard Linux application model. The application is split into kernel space drivers and userspace service shown in Figure.13. The kernel-space drivers provide a bridge between the hardware and the operating system. The userspace side implements a set of client modules adapting driver data into API used by main functionality blocks.

The implementation provides five total clients:

- GPIO Client - Works with the GPIO driver allowing for reading the button input and set the state of the LED.
- Motor Client - Provides the API to configure the PWM functionality to drive the motor. The client allows for setting the period, and duty cycle of PWM set the rotation direction and instantly enable/disable the motor.
- ADC Client - Provides the API for reading the ADC data from the BSP provided IIO interface.
- ADC Compensator - Does not interact with directly with any hardware but provides ADC compensation capabilities based on pre-computed piece-wise linear compensation function.
- Magnetometer client - provides an API reading the magnetometer driver data. The driver features a multi-threaded approach to alleviate potential lock caused by blocking data read at the same time ensuring low-load of the system.

The main application functionality is split into three service modes. The ADC test service allows for obtaining ADC reading for evaluation purposes. When the application is launched in this mode a 5 seconds countdown begins after which the device registers 2^17 raw samples from ADC. The second magnetometer test mode runs in continuous loop printing current magnetometer reading every second. The mode is used to evaluate the correctness of the compass module and the effect of stepper motor on the data reading.

The final Main Service mode is the fully functional implementation of the design goals. The main mode merges all elements and works according to the following algorithm:

1 When a rising edge of a button press is detected the service obtains an initial reading of the magnetometer. Button press starts the single cycle of motor rotation and display of the compass reading results.
2 Next the application obtains the ADC reading then performs compensation. The compensated reading is then normalized to range from -1 to 1.
3 The scaled and normalized ADC value is used to determine the PWM driving frequency. The used motor driver does not support setting variable stepper motor speed through duty cycle; thus the frequency must be used. However, the driver and client pair provides the ability to configure the PWM period and duty cycle.
4 The motor is rotated using the calculated PWM frequency for 0.7 seconds. During the rotation, the LED lights up the signalling device busy.
5 After the rotation finished, another magnetometer reading is performed then both initial and final heading are calculated and displayed.
6 In the last step the service loops back and stays in idle mode until the next button press is detected.

The main service functionality allows for setting the rotation angle using the potentiometer. Then the press of a button starts the cycle. Upon finishing the cycle the initial and final heading angles are displayed.

10. Conclusions

The project concluded with satisfactory results. The final result contains all required hardware elements working as expected. The prototype board features the central Smart4418 embedded system controlling a stepper motor through GPIO and PWM interfaces. The used compass module allows for reading the current device heading. Mounting the compass with the distance of 10cm from the motor allows minimizing the interference of running the motor on the compass reading. The GPIO input/output button and LED combination enables the cycle and indicates device running. The ADC data connected to potentiometer allows for setting up the PWM frequency which determines the rotation angle of the motor.

The user sets the speed and direction of the motor rotation through adjustment of the variable resistor which provides the control signal to ADC. The sampled signal is processed by the ARM system running Linux operating system. A compensation technique applied to raw ADC input corrects the non-ideal characteristics of the ADC.

Signal verification was performed revealing correct signal capture. The power spectrum shows nearly zero harmonics magnitude in the high-frequency region. The results indicate little signal interference and no aliasing. The shape of the power spectrum which resembles a reflection along the midpoint axis shows correctly processed signal from time to frequency domain.

The project utilizes custom developed service software working in user-space and supervising the operation of the input and output devices. Custom character device drivers were written to support user space layer through exposing interfaces necessary to control GPIO pins and PWM resources.

The application structure is divided between kernel-space device drivers and user-space service. The kernel space provides a thin layer of adaptation outputting essential data to user-space layer. The uses space segment features five client modules connecting the raw output of kernel drivers and providing high-level API to be used by main functionality. The application operated in three modes, ADC test, Compass test and Main workflow. The main workflow allows to set the desired rotation angle through the potentiometer then initialize a cycle by pressing the push button.

References

[1] M. Kerrisk, *The Linux programming interface: a Linux and UNIX system programming handbook*. No Starch Press, 2010.

[2] J. Madieu, *Linux Device Drivers Development: Develop customized drivers for embedded Linux*. Packt Publishing Ltd, 2017.

[3] "Ultra compact high performance e-compass3d accelerometer and 3d magnetometer module datasheet." [Online]. Available: https://cdn-shop.adafruit.com/datasheets/LSM303DLHC.PDF

[4] "s5p4418 application processor user's manual 2014." [Online]. Available: http://wiki.friendlyarm.com/wiki/images/a/a7/Pi2_SOC_DS_0.1.pdf

[5] A. V. Oppenheim and R. W. Schafer, *Discrete-time signal processing*. Pearson Education, 2014.

[6] H. Li, "Cmpe242 lecture notes." [Online]. Available: https://github.com/hualili/CMPE242-Embedded-Systems-

[7] FriendlyARM, "Smart4418 sdk." [Online]. Available: http://wiki.friendlyarm.com/wiki/index.php/Smart4418_SDK

[8] ——, "Smart4418." [Online]. Available: http://wiki.friendlyarm.com/wiki/index.php/Smart4418

11. Appendix A: Picture of the prototype board

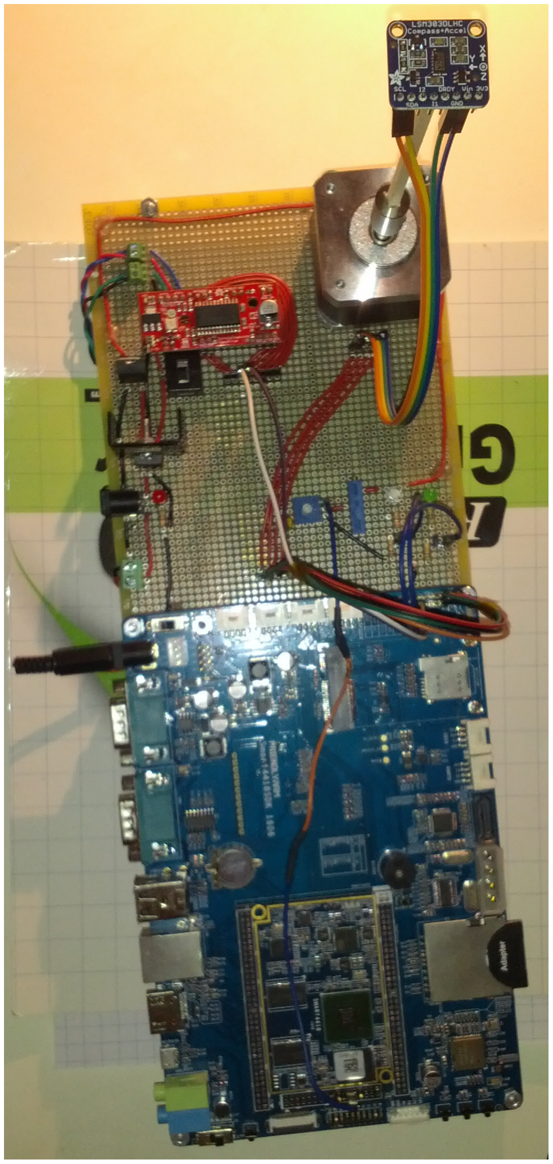

Figure 14. Constructed board

12. Appendix B: Build log from the project1

Listing 7. Project 2 build log

```
../../ee242_sw/prj2/configure --build=x86_64
    -pc-linux-gnu --host=arm-cortexa9-linux-
    gnueabihf CROSS_COMPILE=arm-linux- --
    prefix=/usr

checking for arm-cortexa9-linux-gnueabihf-
    gcc... arm-cortexa9-linux-gnueabihf-gcc
checking whether the C compiler works... yes
checking for C compiler default output file
    name... a.out
checking for suffix of executables...
checking whether we are cross compiling...
    yes
checking for suffix of object files... o
checking whether we are using the GNU C
    compiler... yes
checking whether arm-cortexa9-linux-
    gnueabihf-gcc accepts -g... yes
checking for arm-cortexa9-linux-gnueabihf-
    gcc option to accept ISO C89... none
    needed
checking whether arm-cortexa9-linux-
    gnueabihf-gcc understands -c and -o
    together... yes
checking for atan2 in -lm... yes
checking how to run the C preprocessor...
    arm-cortexa9-linux-gnueabihf-gcc -E
checking for grep that handles long lines
    and -e... /usr/bin/grep
checking for egrep... /usr/bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking stdio.h usability... yes
checking stdio.h presence... yes
checking for stdio.h... yes
checking for stdlib.h... (cached) yes
checking for stdint.h... (cached) yes
checking for unistd.h... (cached) yes
checking fcntl.h usability... yes
checking fcntl.h presence... yes
checking for fcntl.h... yes
checking getopt.h usability... yes
checking getopt.h presence... yes
checking for getopt.h... yes
checking errno.h usability... yes
checking errno.h presence... yes
checking for errno.h... yes
checking for string.h... (cached) yes
checking syslog.h usability... yes
checking syslog.h presence... yes
checking for syslog.h... yes
checking signal.h usability... yes
checking signal.h presence... yes
checking for signal.h... yes
checking sys/mman.h usability... yes
checking sys/mman.h presence... yes
checking for sys/mman.h... yes
checking for sys/stat.h... (cached) yes
checking for inttypes.h... (cached) yes
checking limits.h usability... yes
checking limits.h presence... yes
checking for limits.h... yes
checking stddef.h usability... yes
checking stddef.h presence... yes
checking for stddef.h... yes
checking for string.h... (cached) yes
checking sys/time.h usability... yes
checking sys/time.h presence... yes
checking for sys/time.h... yes
checking sys/timeb.h usability... yes
checking sys/timeb.h presence... yes
checking for sys/timeb.h... yes
checking for inline... inline
checking for pid_t... yes
checking for size_t... yes
checking for ssize_t... yes
checking for uint32_t... yes
checking vfork.h usability... no
checking vfork.h presence... no
checking for vfork.h... no
checking for fork... yes
checking for vfork... yes
checking for working fork... cross
checking for working vfork... (cached) yes
checking for stdlib.h... (cached) yes
checking for GNU libc compatible malloc...
    no
checking for ftime... yes
checking for gettimeofday... yes
checking for memset... yes
checking for pow... yes
checking for a BSD-compatible install... /
    usr/bin/install -c
checking whether build environment is sane
    ... yes
checking for arm-cortexa9-linux-gnueabihf-
    strip... arm-cortexa9-linux-gnueabihf-
    strip
checking for a thread-safe mkdir -p... /usr/
    bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports the include
    directive... yes (GNU style)
checking whether make supports nested
    variables... yes
checking dependency style of arm-cortexa9-
    linux-gnueabihf-gcc... gcc3
checking that generated files are newer than
     configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating scripts/Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands
```

```
checking that generated files are newer than
     configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating scripts/Makefile
config.status: creating config.h
config.status: config.h is unchanged
config.status: executing depfiles commands

make  all-recursive
make[1]: Entering directory '/opt/
    FriendlyARM/build_dir/prj2'
Making all in src
make[2]: Entering directory '/opt/
    FriendlyARM/build_dir/prj2/src'
arm-cortexa9-linux-gnueabihf-gcc -
    DHAVE_CONFIG_H -I. -I../../../ee242_sw/
    prj2/src -I..      -g -O2 -MT main.o -MD -
    MP -MF .deps/main.Tpo -c -o main.o
    ../../../ee242_sw/prj2/src/main.c
mv -f .deps/main.Tpo .deps/main.Po
arm-cortexa9-linux-gnueabihf-gcc -
    DHAVE_CONFIG_H -I. -I../../../ee242_sw/
    prj2/src -I..      -g -O2 -MT service.o -
    MD -MP -MF .deps/service.Tpo -c -o
    service.o ../../../ee242_sw/prj2/src/
    service.c
mv -f .deps/service.Tpo .deps/service.Po
arm-cortexa9-linux-gnueabihf-gcc -
    DHAVE_CONFIG_H -I. -I../../../ee242_sw/
    prj2/src -I..      -g -O2 -MT adc_client.o
     -MD -MP -MF .deps/adc_client.Tpo -c -o
    adc_client.o ../../../ee242_sw/prj2/src/
    adc_client.c
mv -f .deps/adc_client.Tpo .deps/adc_client.
    Po
arm-cortexa9-linux-gnueabihf-gcc -
    DHAVE_CONFIG_H -I. -I../../../ee242_sw/
    prj2/src -I..      -g -O2 -MT compensator.
    o -MD -MP -MF .deps/compensator.Tpo -c -o
     compensator.o ../../../ee242_sw/prj2/src
    /compensator.c
mv -f .deps/compensator.Tpo .deps/
    compensator.Po
arm-cortexa9-linux-gnueabihf-gcc -
    DHAVE_CONFIG_H -I. -I../../../ee242_sw/
    prj2/src -I..      -g -O2 -MT motor_client
    .o -MD -MP -MF .deps/motor_client.Tpo -c
    -o motor_client.o ../../../ee242_sw/prj2/
    src/motor_client.c
mv -f .deps/motor_client.Tpo .deps/
    motor_client.Po
arm-cortexa9-linux-gnueabihf-gcc -
    DHAVE_CONFIG_H -I. -I../../../ee242_sw/
    prj2/src -I..      -g -O2 -MT adc_sampler.
    o -MD -MP -MF .deps/adc_sampler.Tpo -c -o
     adc_sampler.o ../../../ee242_sw/prj2/src
    /adc_sampler.c
mv -f .deps/adc_sampler.Tpo .deps/
    adc_sampler.Po
arm-cortexa9-linux-gnueabihf-gcc -
    DHAVE_CONFIG_H -I. -I../../../ee242_sw/
    prj2/src -I..      -g -O2 -MT gpio_client.
    o -MD -MP -MF .deps/gpio_client.Tpo -c -o
     gpio_client.o ../../../ee242_sw/prj2/src
    /gpio_client.c
mv -f .deps/gpio_client.Tpo .deps/
    gpio_client.Po
arm-cortexa9-linux-gnueabihf-gcc -
    DHAVE_CONFIG_H -I. -I../../../ee242_sw/
    prj2/src -I..      -g -O2 -MT mag_client.o
     -MD -MP -MF .deps/mag_client.Tpo -c -o
    mag_client.o ../../../ee242_sw/prj2/src/
    mag_client.c
mv -f .deps/mag_client.Tpo .deps/mag_client.
    Po
arm-cortexa9-linux-gnueabihf-gcc  -g -O2   -
    o ee242_prj2 main.o service.o adc_client.
    o compensator.o motor_client.o
    adc_sampler.o gpio_client.o mag_client.o
    -lm -lpthread -lm
make[2]: Leaving directory '/opt/FriendlyARM
    /build_dir/prj2/src'
Making all in scripts
make[2]: Entering directory '/opt/
    FriendlyARM/build_dir/prj2/scripts'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/opt/FriendlyARM
    /build_dir/prj2/scripts'
make[2]: Entering directory '/opt/
    FriendlyARM/build_dir/prj2'
make[2]: Leaving directory '/opt/FriendlyARM
    /build_dir/prj2'
make[1]: Leaving directory '/opt/FriendlyARM
    /build_dir/prj2'
```