Project Report: M2M Motor rig project

**Document information**

|  |  |
| --- | --- |
| Subject | EEE8089 M2M Technology IoT |
| Authors | Elena Quesada |
| Keywords | M2M, Raspberry, Routing, Motor, LAN, PID, SSH |

**Table of contents**

[1. DESCRIPTION OF THE PROJECT 3](#_Toc39494353)

[1.1. CONFIGURATION OF THE NETWORK 4](#_Toc39494354)

[1.2. NAT CONFIGURATION 7](#_Toc39494355)

[1.3. SECURE SHELL LAYER 7](#_Toc39494356)

[1.4. MOTOR RIG 8](#_Toc39494357)

[2. CONCLUSIONS 13](#_Toc39494358)

[ANNEX 14](#_Toc39494359)

1. DESCRIPTION OF THE PROJECT

The aim of the project is to build a safety Machine to machine (M2M) network to control remotely the spinning speed and the direction of a motor rig.

The M2M network will be defined following OSI model to implement each of the layer related in the network as it is seen on Figure 1. TCP/IP over Ethernet will be configured and will support the secure shell, allowing the application to transmit in a secure way. The application layer will be the design of the motor rig controller. The different layers can be seen in the figure.



Figure 1. OSI model analogy

Additionally, WIFI protocol will be used to develop the extend the functionality of the network. 4 different raspberry PI will be used as nodes to build, configure and implement the network together with a switch to connect 3 different private LANs over ethernet connection.

The application layer will design the motor rig controller. The direction and the spin speed of the motor will be commanded and send to the motor using the UART. Reading from a built-in encoder the application will get some feedback related to the real speed and direction of the motor and acts consequently. The implementation of a PID will allow correcting the error got from the motor due to external factors. The motor rig con used is a 24V DC motor provided by the university together with an already programmed protoboard to control it. The specification of this protoboard has to be consider when the motor is going to be used.

Although no functional application has been built over the motor controller, different functionalities can be implemented. For example, once it is possible to control the speed or frequency of the motor, associating each frequency with a note, different music songs can be played. This functionality was not an intended part of the project, but is good to look forward and see the usability of the project.

This report presents the SW implementation of the application layer and the network configuration that has been carried out during the lab sessions. Thus, if it was not possible to finish some of the stages of the project due to the current situation, a theorical approach is given instead.

* 1. CONFIGURATION OF THE NETWORK

Figure 2 shows the layout of the network. As it can be seen 4 nodes are connected between them using 3 different private LANs. Even if all of them are connected to the same switch as a matter of resources, it is not going to be able to reroute data for example from node 1 to node 4. This way, a message from a node will have to go through all the different LANs in the middle until its destination. This configuration is intended to reflect a more complex network where different LANs are needed because of the coverage distance.

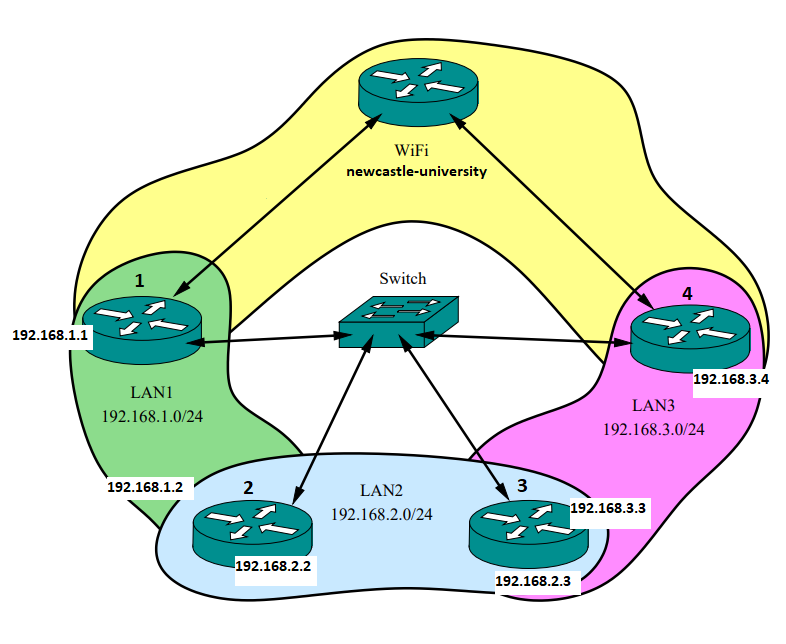


Figure 2. Network configuration

Same reason can explain the configuration of WIFI connection, although it is not explicitly used in this project. Newcastle university WIFI is configured allowing to send data to the network. This data could have been displayed in the cloud and used later on for further analysis.

The configuration of the network is presented in detail in the following sections.

* + 1. WIFI CONFIGURATION

Edge nodes will configurate WIFI connection and they will work as gateways in case any middle node needs to send data to the cloud. Since Newcastle WIFI connection is used, which is configure as a secure network, some specific configurations have to be made as follow.

Admin privileges are needed and ***wpa\_supplicant*** file has to be modified as shown in Figure 3. This configuration has to be done just once.

>>cd **/etc/wpa\_supplicant**

**>>sudo leafpad wpa\_supplicant.conf**

|  |
| --- |
| **network={**  **ssid="newcastle-university"**  **key\_mgmt=WPA-EAP**  **eap=TTLS**  **identity="your\_login\_name"**  **password="your\_password"**  **phase2="auth=MSCHAPv2"** **}** |

Figure 3.wpa\_supplicant file configuration

Node 1 and node 4 will configurate WIFI and work as a gateway for node 2 and 3. Middle nodes could have got WIFI connection too, but the system version of the raspberry Pi were not compatible with the configuration of several IPs id in the nodes without making some other changes, and since it was not part of the aim of the project, it was not made.

Network file has to be also configured in order to set wlan0 as it is shown in Figure 4:

>>cd **/etc/network**

**>>sudo leafpad interfaces**

|  |
| --- |
| **# Configure WIFI**  **allow-hotplug wlan0 iface wlan0 inet manual wpa-conf /etc/wpa\_supplicant/wpa\_supplicant.conf** |

Figure 4. Interface file configuration for WIFI

* + 1. LAN CONFIGURATION

Three different private LANs are configured, each of them with its on IP. Each node will work as a gateway for other LAN so each node will have two different IP configuration. Figure 2 shows the distributions of the LAN and the IP given to each of them.

In order to configure each of the LANs, the interface file has to be modified. An example of the configuration needed is shown in Figure 5 where LAN 1 and LAN2 are configured with their Ips for node 2. Each node will configure two interfaces: either two LAN or one LAN and the WIFI interface. Both interfaces exist together in the same node allowing the data routing from one node to other:

>>cd **/etc/network**

**>>sudo leafpad interfaces**

|  |
| --- |
| **# Configure LAN 1**  auto eth0  allow-hotplug eth0  iface eth0  inet static address 192.168.1.2  netmask 255.255.255.0  **# Configure LAN 2**  auto eth0:1  allow-hotplug eth0:1  iface eth0:1  inet static address 192.168.2.2  netmask 255.255.255.0 |

Figure 5. Interface file configuration configuring LAN 1 and LAN 2 for node 2

The configuration of the interface file has to be made each time the node is rebooted either manually or using a script.

* + 1. ROUTING

As it was said, middle nodes are going to work as routers. Thus, IPv4 routing must be enable in the raspberry PI. Furthermore, since all of the nodes are connected to the same switch but we do not want Linux to update its ARP table, it is needed to disable redirecting accepts and sends. In order to do that, sysctl.conf file has to be modified as it is shown on Figure 6. This way the messages will be routed from the node 1 to the node 4, going through all the nodes in the middle:

**>>**cd **/etc**

**>>sudo leafpad** sysctl.conf

|  |
| --- |
| #enable routing  net.ipv4.ip\_forward=1  # inhibit accept redirects  net.ipv4.conf.all.accept\_redirects = 0  #inhibits send redirects  net.ipv4.conf.all.send\_redirects = 0 |

Figure 6. sysctl.conf file

When a message has as destination a node that is not in the LAN, it has to be routed using the gateways. The gateways will be configured on each node in order to create the ARP tables according to the configuration. For example, node 4 will configures its gateway of LAN3 as follows:

>>**route add -net 192.168.30 netmask 255.255.255.0 gw 192.168.2.3**

**Thus, the gateway of the LAN3 for node 4 is the IP related with the LAN 2 for the node 3.** ARP tables needs to be fulfilled each time the node is rebooted either manually or using a script.

* 1. NAT CONFIGURATION

Node 1 and 4 are not only going to work as gateways for the internet connection, they are also masquerade other nodes IP within the LANs to work as the same node. This is useful when several nodes or computer are connecting using even different routers to connect them but it is wanted all of them work as the same device.

Masquerade functionality has to be enabled in the node as follows:

**>>iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o wlan0 -j MASQUERADE**

**This behaviour is implemented in order to explore all the functionalities allowed but it is not going to be used within the project since no any big data application has been implemented.**

* 1. SECURE SHELL LAYER

The secure shell (SSH) protocol is a method for secure remote login from M2M. It works as the safety and presentation layer of the network and guaranteeing to establish a secure session and transmitted in a controlled and reliable way protecting the communications security and integrity with strong encryption.

SSH works as a client-server model, where the client is responsible for establishing the connection with the server using a hand-shake algorithm to share public keys and open a secure channel (see Figure 7). After the setup phase the SSH protocol uses strong symmetric encryption and hashing algorithms to ensure the privacy and integrity of the data that is exchanged between the client and server.

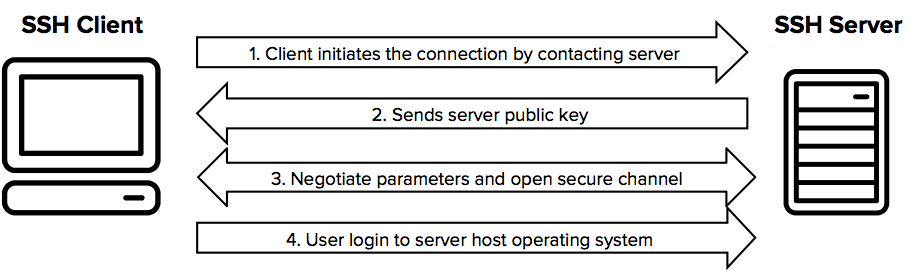


Figure 7. SSH establish session hand-shake

The SSH communication is one-way direction, and since the data in the network will be bidirectional two session has to be established among nodes.

For the configuration of the SSH, each node working as a client will create some private and public keys, and will share them with other nodes. Once the session is established, the client will work as if it were the server node. Notice that if the client and server node are not in the same LAN, lower layers will reroute allowing the session to be establishes.

Client node will generate private and public keys and share them with the server as follows:

**>>ssh-keygen**

**>>ssh-copy 192.168.X.X**

* 1. MOTOR RIG

Motor rig controller allows being aware of the real speed and direction of a 24 DC motor and correct it when it is not the expected one. When a motor is spinning different external factors like external resistance can affect in the process and slow down or speed up the real spinning of the motor. The motor controller has been developed using two different SW process. Each process can be run in different nodes since the communication network that has been built enables to send data in a safe way.

Executing each of the processes from different emplacement adds some delay in the system but introduce so much the capability to that can be also used for critical systems. The layout of the system has been tested using the same node to control the motor, and then adding the safety network. This way the delay can be characterized and introduced in the system in a controlled scenario.



Figure 8. System architecture

The encoder SW process will read data from the encoder placed on the protoboard to get the current state of the motor. On the other hand, regulator SW process commands the value and direction that the motor is meant to spin and implements a PID to correct the error: The final output is written in the motor. Since both processes are placed in different nodes, secure shell is used to transmit the data. The communication between the regulator and the motor is made using the UART of the node, and the encoder data is read using general I/O. Figure 8 shows the system architecture of the whole system.

In order to share data between processes, pipelining is used. If the processes are run in different nodes the secure shell guarantee the secure transmission between the nodes, adding a delay that has to be estimated. The encoder process send as an output in mHz is sent as an input to the regulator because it is used to calculated the error. In the first case, files are defined to **store the data as follows:**

**>> cat FIFO\_NODE1 | ssh 192.168.x.x “>” FIFO\_NODE4**

If the processes are run in the same node, **stdin** and **stdout** are used

The following sections are going to present each process in-depth.

* + 1. Encoder

The encoder SW process will use the value from the encoder as an input and will calculate the mHz output.

In order to do that, two threads are used since some of the actions the process has to do are blocking, which means the process has to wait in a certain point until a task end.

First thread will count the number of pulses that are detected by the photodiodes while the second thread will set a timer. When the timer is up, the second thread reads the value of the counter to calculate the speed in mHz and resets it. Since the counter is used by the two threads, a mutex is needed in order to define a critical section and avoid simultaneous access to the variable. The flow chart of the encoder SW process is shown in Figure 9. The two are created and a join point is used to avoid main function ends without the execution of the threads.

The global variables used by the encoder SW process are:

* u32\_encoder\_counter: used as a counter of the photodiode pulses. This variable is used by the two threads. Hence, a critical section will be defined to access this variable.
* u8\_direction: direction of spinning. It will value 0 if the motor is spinning in one way, and 1 if it is spinning in the other.
* mutex: used as a semaphore for the critical section.



Figure 9. Encoder SW process

* + - 1. Thread 1: Counter

Thread 1 will define a while loop to keep executing itself continuously, but first GPIO has to be initialized to use GPIO9 and GPIO10.

The thread will use poll() function to check if a rising pulse of GPIO9 has occurred. This function is blocking, so the process will wait at this point until poll() function finishes When it happens, and IRQ unblocked the process. Then, GPIO10 is checked in order to know the direction of the motor. If the value of GPIO10 has changed, the variable u8\_direction is updated. This is made the sooner the IRQ sets off to avoid misreading, but an additional filter is implemented in the second thread because some error readings were still happening. Mutex is locked to increase the counter, and unlocked right after it. The flow chart of the thread is shown in Figure 10. Yellow colour shows the critical section



Figure 10. Thread 1 flow

* + - * 1. GPIO

In order to be able to know the state of the motor, the motor rig includes an encoder that gives information about the speed and direction. The encoder is made using a wheel connected to the motor with 500 holes uniformly distributed, one LED and two photodiodes. The light from the LED will go through the holes while the wheel is spinning. Thus, the photodiodes generate a pulse every time the light reaches the photodiodes as it is seen in Figure 11. The direction of the wheel can be calculated looking at which of the photodiode rises the first one.

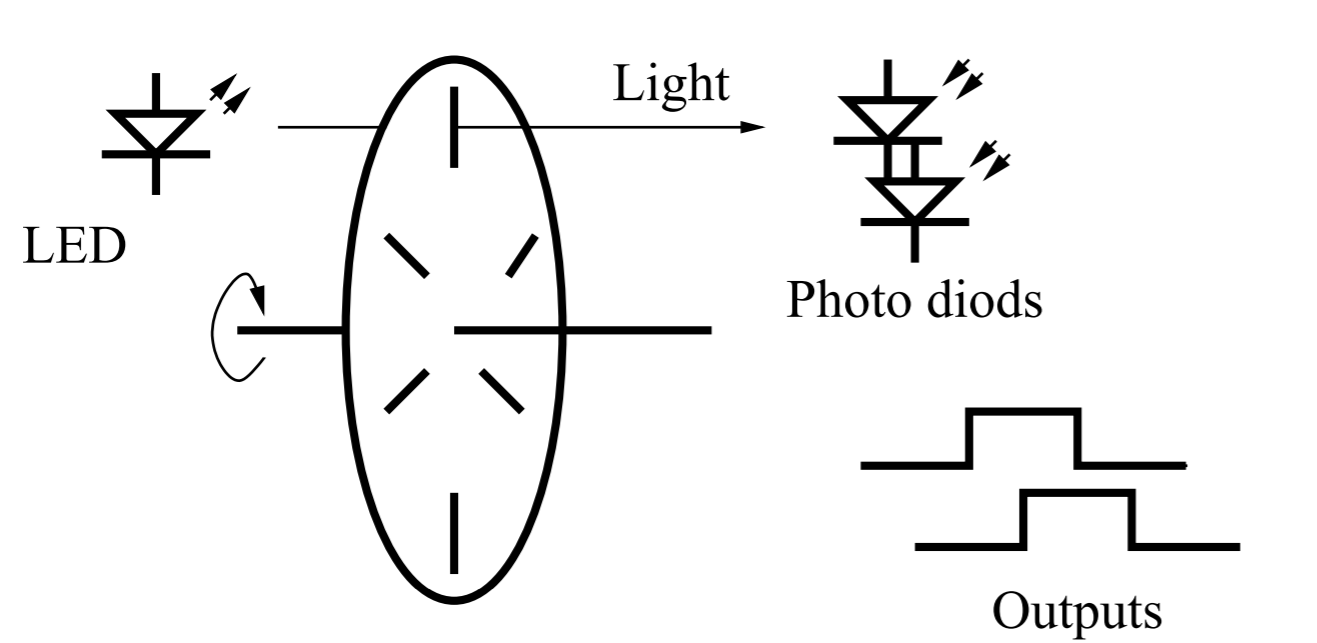


Figure 11. HW encoder

Each photodiode is connected to a general input: the first photodiode is connected to the GPIO 9 and will be used to calculate the speed while second photodiode, connected to GPIO10, will get the spinning direction. The value of the inputs will be saved in different files that is configured during init\_encoder() to be configure as follows:

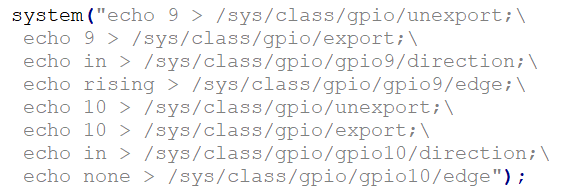


Figure 12. GPIO configuration

Each GPIO will configure the type of GPIO as an input and the edge detection. GPIO9 will use and IRQ when the pulse risen to count the number of pulses, so the edge detection is configured as rising. GPIO10 will not use any IRQ so there is no need to configure edge detection, but it will read the value of the GPIO from /sys/class/gpio/gpio10/value.

* + - 1. Thread 2: Timer

Another thread is needed to fix a period of time as a sample and be able to get the mHz of the encoder. The time is defined by TIMER constant and its value has to be carefully since a small time will not allow to get correct values when the motor is turning slowly and a long time will lead in the addition to more delay. In this case we started with a timer TIMER= 20 ms.

The thread will be executed continuously in a while loop. The function usleep() acts like a blocking function until the TIMER is up. Then, using the mutex, the value of the counter is read and reset. The value read means the number of pulses detected within the TIMER. In order to translate that into mHz the following equation are applied bearing in mind that the encoder wheel has 500 holes uniformly distributed among the whole circle.:

Then, the speed in mHz is calculated as follows:

The output is printed in the stdout using an printf(). It is important to fflush() the buffer to make sure the values are written. Figure 13 shows the flow of the process. Yellow colours show the critical section.



Figure 13. Thread 2 flow

The filter functions shown in the figure is added after several tests at the labs that were showing errors due to misreading in the direction. This filter is explained in the following section.

* + - * 1. Filter

Since the speed of the motor varies and GPIO read takes time, if the process does not read the value of GPIO10 on time, it is possible to get misreading in the direction of the speed. For this reason, a filter was used to avoid these misreading. The filter was implemented in the second threads since the execution frequency of this one is much slower (ever TIMER instead of just a few microseconds) and then, the whole process is more efficient.

The filter is implemented using a counter. When the direction changes, it has to do it at least during 5 consecutives cycles. Otherwise, the change will not be considered. It adds a delay in the detection of a changing direction that is directly affected by the TIMER and the filter itself.

* + 1. Regulator
       1. Speed conversion
       2. PID

CONCLUSIONS

ANNEX

/\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*! **\file**

\*

\*\* PROJECT: M2M.2020

\*\*

\*\* NAME: encoder.c

\*\*

\*\* AUTHOR: Group 4

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <assert.h>

#include <poll.h>

#include <pthread.h>

#include "encoder.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Global variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

s\_encoder\_data vg\_encounter\_data **=**

**{**

/\* Global variable initialiazation \*/

0**,**

2**,**

PTHREAD\_MUTEX\_INITIALIZER**,**

**};**

int main**(**void**)**

**{**

/\* Local variable definition \*/

pthread\_t thread1**,** thread2**;**

int iret1**,** iret2**;**

/\* Create two threads: encoder and polling GPIO \*/

iret1 **=** pthread\_create**(&**thread1**,** **NULL,** **(**void**\*)**encoder**,** **NULL);**

iret2 **=** pthread\_create**(&**thread2**,** **NULL,** **(**void**\*)**poll\_irq**,** **NULL);**

/\* Wait until threads are complete before main continues \*/

pthread\_join**(**thread1**,**0**);**

pthread\_join**(**thread2**,**0**);**

**return** EXIT\_SUCCESS**;**

**}**/\* main \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* FUNCTION: init\_encoder

\*\* INPUTS:

\*\* None

\*\* OUTPUTS:

\*\* None

\*\* COMMENTS:

\*\* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

\*\* Initializate GPIO for the encoder

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void init\_encoder**(**void**)**

**{**

/\* Configure PIN 9 as GPIO associate to the IRQ and PIN9 to check spinning

\* direction \*/

system**(**"echo 9 > /sys/class/gpio/unexport;\

echo 9 > /sys/class/gpio/export;\

echo in > /sys/class/gpio/gpio9/direction;\

echo rising > /sys/class/gpio/gpio9/edge;\

echo 10 > /sys/class/gpio/unexport;\

echo 10 > /sys/class/gpio/export;\

echo in > /sys/class/gpio/gpio10/direction;\

echo none > /sys/class/gpio/gpio10/edge"**);**

**return;**

**}**/\* init\_encoder \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* FUNCTION: encoder

\*\* INPUTS:

\*\* ps\_data: encoder data

\*\* OUTPUTS:

\*\* None

\*\* COMMENTS:

\*\* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

\*\* Set a timer and check the number of the encoder counter

\*\* when the timer is up

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void**\*** encoder**(**void**)**

**{**

unsigned long int u32\_total\_cycles **=** 0**;**

int i32\_cycles\_mhz **=** 0**;**

int fd2 **=** 0**;**

int direction **=** 1**;**

int counter **=** 0**;**

//printf("Encoder\n");

**while(**1**)**

**{**

u32\_total\_cycles **=** 0**;**

usleep**(**TIMER**);**

/\* get the mutex \*/

pthread\_mutex\_lock **(&**vg\_encounter\_data**.**mutex**);**

/\* Read the counter \*/

u32\_total\_cycles **=** vg\_encounter\_data**.**u32\_encoder\_counter**;**

/\* Reset the counter \*/

vg\_encounter\_data**.**u32\_encoder\_counter **=** 0**;**

/\* Unlocked the mutex \*/

pthread\_mutex\_unlock **(&**vg\_encounter\_data**.**mutex**);**

**if(**direction **!=** vg\_encounter\_data**.**u8\_direction**)**

**{**

counter**++;**

**if(**counter **>** 5**)**

**{**

direction **=** **(**direction **+** 1**)%**2**;**

**}**

**}**

**else**

**{**

counter **=** 0**;**

**}**

**if(**direction **==** 1**)**

**{**

/\* Compute result \*/

i32\_cycles\_mhz**=**u32\_total\_cycles**\***100**;**

**}**

**else**

**{**

/\* Compute result \*/

i32\_cycles\_mhz**=** **-(**u32\_total\_cycles**\***100**);**

**}**

/\* Write result in output file \*/

//printf("%ld\n", vg\_encounter\_data.u8\_direction);

/\* Write result in output file \*/

printf**(**"%d\n"**,** i32\_cycles\_mhz**);**

fflush**(**stdout**);**

**}**

**}**/\* encoder \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* FUNCTION: poll\_irq

\*\* INPUTS:

\*\* ps\_data: encoder data

\*\* OUTPUTS:

\*\* None

\*\* COMMENTS:

\*\* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

\*\* Configure the IRQ associate to one of the diods and polling

\*\* it incrementing a counter

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void**\*** poll\_irq**(**void**)**

**{**

/\* Local variable definition \*/

struct pollfd pfd0**;**

unsigned char u8\_value **=** 0**;**

int fd1 **=** 0**;**

/\* Configuring GPIO \*/

init\_encoder**();**

/\* Configuring POLL events \*/

pfd0**.**events **=** POLLPRI **|** POLLERR**;**

pfd0**.**fd **=** open**(**"/sys/class/gpio/gpio9/value"**,** O\_RDONLY**|**O\_NONBLOCK**);**

fd1 **=** open**(**"/sys/class/gpio/gpio10/value"**,** O\_RDONLY **|** O\_NONBLOCK**);**

//printf("POLL IRQ\n");

**while(**1**)**

**{**

/\* Read from the beginning (offset 0 ) \*/

lseek**(**pfd0**.**fd**,** 0**,** SEEK\_SET**);**

/\* Empty the read buffer \*/

read **(**pfd0**.**fd**,** **&**u8\_value**,** 1**);**

/\* Wait until the next interruption. This function blocks the process \*/

poll**(&**pfd0**,** 1**,** 1000**);**

/\* Read PIN10 for direction \*/

lseek**(**fd1**,** 0**,** SEEK\_SET**);**

/\* read the value \*/

read **(**fd1**,** **&**u8\_value**,** 1**);**

/\* Get the mutex \*/

pthread\_mutex\_lock **(&**vg\_encounter\_data**.**mutex**);**

/\* Increase the counter \*/

vg\_encounter\_data**.**u32\_encoder\_counter**++;**

/\*Release the mutex \*/

pthread\_mutex\_unlock **(&**vg\_encounter\_data**.**mutex**);**

/\* Check if the direction has changed \*/

**if(**vg\_encounter\_data**.**u8\_direction **!=** **(**u8\_value**%**2**))**

**{**

vg\_encounter\_data**.**u8\_direction **=** u8\_value**%**2**;**

**}**

**}**

**}**/\* poll \*/

/\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*! **\file**

\*

\*\* PROJECT: M2M.2020

\*\*

\*\* NAME: encoder.h

\*\*

\*\* AUTHOR: Group 4

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <pthread.h>

/\* ! Sampling timer in us \*/

#define TIMER (unsigned long int) 20000

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Structure declaration

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**typedef** struct \_s\_encoder\_data

**{**

/\* Flash counter \*/

unsigned long int u32\_encoder\_counter**;**

/\* Spinning direction \*/

unsigned char u8\_direction**;**

/\* Mutex \*/

pthread\_mutex\_t mutex**;**

**}**s\_encoder\_data**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function declaration

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*! Encoder function. Set a timer and check the number of the encoder counter

\* when the timer is up

ps\_data: input data \*/

void**\*** encoder**(**void**);**

/\*! Poll function. Configure the IRQ associate to one of the diods and polling

\* it incrementing a counter

\* ps\_data: input data \*/

void**\*** poll\_irq**(**void**);**

/\*! Initializate GPIO for the encoder \*/

void init\_encoder**(**void**);**

/\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*! **\file**

\*

\*\* PROJECT: M2M.2020

\*\*

\*\* NAME: regulator

\*\*

\*\* AUTHOR: Group 4

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <assert.h>

#include "regulator.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Global variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main**(**void**)**

**{**

/\* Local variable definition \*/

int i32\_read\_value **=** 0**;**

float kp **=** 4**;**

float ki **=** 0.2**;**

float kd **=** 0**;**

int last\_error **=** 0**;**

int error **=** 0**;**

long long int pid **=**0**;**

int input\_Mhz **=** 6600**;**

float integral **=** 0**;**

float derivative **=** 0**;**

int output **=** 0**;**

//int integral = 0;

output **=** f\_get\_motor\_values**(**input\_Mhz **+** pid**);**

printf**(**"%d 1\\"**,** output**);**

fflush**(**stdout**);**

**while(**1**)**

**{**

scanf**(**"%d"**,&**i32\_read\_value**);**

fprintf**(**stderr**,** "Read value %d\n"**,** i32\_read\_value**);**

/\* PID is used to calculate the error \*/

error **=** input\_Mhz **-** i32\_read\_value**;**

fprintf**(**stderr**,** "Error: %d\n"**,** error**);**

integral **=** **(**integral **+** error**);**

derivative **=** **(**error **-** last\_error**);**

last\_error **=** error**;**

pid **=** **(**int**)(**kp **\*** error **+** ki**\***integral **+** kd**\***derivative**);**

fprintf**(**stderr**,** "PID: %d\n"**,** pid**);**

/\* Transform the value the motor to get the value in the range,

\* adjusting the value according to the PID result \*/

output **=** **(**int**)**f\_get\_motor\_values**(**pid**);**

/\* If output is out of range, it is adapted \*/

**if(**output **>** 1237**)**

**{**

output **=** 1236**;**

**}**

**else** **if(**output **<** 237**)**

**{**

output **=** 238**;**

**}**

fprintf**(**stderr**,** "Output: %d\n"**,** output**);**

/\* Print the output for the motor \*/

printf**(**"%d 1\\"**,** output**);**

fflush**(**stdout**);**

**}**

**return** EXIT\_SUCCESS**;**

**}**/\* main \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* FUNCTION: f\_get\_motor\_values

\*\* INPUTS:

\*\* input\_value: MHz

\*\* OUTPUTS:

\*\* int: motor value

\*\* COMMENTS:

\*\* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

\*\* Calculate the output for the motor according to the MHz

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int f\_get\_motor\_values **(**long long int input\_value**)**

**{**

int motor\_value **=** 0**;**

**if(**input\_value**>**0**)**

**{**

motor\_value**=(**input\_value**+**15262**)/**19.8**;**

**}**

**else** **if(**input\_value**<**0**)**

**{**

motor\_value**=(**input\_value**+**7870**)/**10**;**

**}**

**else**

**{**

motor\_value**=**737**;**

**}**

**return** motor\_value**;**

**}**/\* f\_get\_motor\_values \*/

/\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*! **\file**

\*

\*\* PROJECT: M2M.2020

\*\*

\*\* NAME: regulator.h

\*\*

\*\* AUTHOR: Group 4

\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function declaration

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*! Motor function. Calculate the output for the motor according to the MHz

input\_value: input data \*/

int f\_get\_motor\_values **(**long long int input\_value**);**