ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE - DIN

APPLICATION NOTES

for

SOFTWARE DEVELOPMENT OF AN ETHERCAT BASED PROTOTYPE

Grab Cable Robot

Developers:
Ing. Edoardo Idà

Supervisor:
Prof. Ing. Marco Carricato

Version 0.0 - 22/01/2018

Abstract

These application notes describe in details the steps necessary to:

- build a real-time (*rt* in the following) capable linux based pc;
- install and set-up the ethercat master library provided by the Etherlab software component;
- program and run a simple *rt* application where the *rt* pc is connected to a relatively minimal ethercat slave device: an EasyCAT;
- describe in detail how the Grab prototype is programmed so that its software can be used and updated with ease.

Contents

Chapter 1

Building the *rt* linux pc

This part ensures that the master controller of the robot can be used as a regular industrial pc: the pc has infact hard real-time capabilities and a full control of the peripherals of the machine.

1.1 Some hardware tips

You will need:

- A Pc case with at least one usb 2.0 port;
- A power supply unit: better use low power ones with high efficiency (more stable voltage and faster response);
- A 6th or 7th Intel Cpu;
- Any recent standard or micro ATX motherboard compatible with the processor: recent test demonstrated that the most simple choice would be one with and integrated network card using e1000e driver, but it is not mandatory;
- At least 16 Gb of the fastest ram compatible with the Cpu and motherboard;
- a Pci-e network card with at least one ethernet port. It is important that at least one between the integrated network card and the Pci-e one is based on one these drivers: 8139too, e1000, e100, r8169, e1000e. Usually the modern Intel integrated network card are based on e1000e but it is necessary to check. If an integrated network card with e1000e driver is embedded onto the motherboard, a r8169 lan card must be chosen.
- It is STRONGLY suggested to use the r8169 lan card as internet port and the e1000e one as the ethercat port because of some minor compatibility issue with newer linux kernel and etherlab software, that we will be installing shortly.

It is best to keep the configuration of the Pc to its minimum, not adding video card or any other Pci card, if it is not crucial to the application.

1.2 Installing Linux

First of all a basic Debian based operating system must be installed on the machine. In this notes everything will be related to the distribution *Ubuntu 16.04.3 LTS*.

- go to https://www.ubuntu-it.org/download and download Ubuntu 16.04.3 LTS;
- create a bootable usb stick with it (use Rufus for Windows, for example);
- install the operating system alongside Microsoft Windows (it is necessary because the majority of drive systems has to be set up in a Windows based environment);
- be sure to leave twice the ram space as swap area and to have only one additional partition as root (/);
- download all the necessary upload during installation: better having wired ethernet connection

Once Ubuntu is up and running, a real-time kernel must be installed.

1.3 Installing the real-time kernel

Some kernel version cannot be installed, depending on the linux distribution one is using. For example, with Ubuntu 16.04.3 LTS only 4.x kernel can be used. One of the lastest stable release of the *rt* kernel is the 4.13.13 so this one will be used in the following.

- go to https://www.kernel.org/pub/linux/kernel/v4.x/ and download linux-4.13.13.tar.xz;
- go to https://www.kernel.org/pub/linux/kernel/projects/rt/4.13/ and download patch-4.13.13-rt5.patch.xz (it may be in older/);
- open a terminal and execute the following commands:
  - cd Download
  - tar xvfj linux-4.13.13.tar.xz
  - cd linux-4.13.13
  - xzcat ../patch-4.13.13-rt5.patch.xz | patch -p1
  - cp /boot/config-$(uname -r) .config
  - yes "" | make oldconfig
  - sudo apt-get install libncurses-dev libssl-dev
  - make menuconfig -> Processor type and features -> Preemption Model -> Fully Preemptible Kernel (RT) and save
  - make -j4 (the number depends on the number of core available in the Cpu, the higher the less compile time)
  - make modules -j4 (same as above)

- sudo make modules_install
- sudo make install
- sudo update-grub

Now you should be able to boot without problems with your newly installed real-time kernel. As far as the user is concerned, he will sometimes experience a little bit of lag the monitor interface but that is completely normal. Reboot with the real-time kernel using advanced option in the grub at startup. To test the kernel and gain an idea of how a real-time application is set up https://wiki.linuxfoundation.org/realtime/start can be consulted.

Chapter 2

Installing and setting up Ethercat Master

At this stage the ethercat master application must be installed. Most of this walk-through guide has been developed following the instruction that can be found at https://sourceforge.net/u/uecasm/etherlab-patches/ci/tip/tree/ .

2.1 Getting the source code and building

Open a terminal and type:

- sudo -s
- apt install mercurial
- gedit /etc/mercurial/hgrc/hgrc and add the following lines

```
[extensions]
mq =
```

- cd /usr/src
- hg clone -u 33b922ec1871 http://hg.code.sf.net/p/etherlabmaster/code etherlab
- hg clone http://hg.code.sf.net/u/uecasm/etherlab-patches etherlab/.hg/patches
- cd etherlab
- hg qpush -a

At this point we retrieved the source code of ethercat master and patched so that is compatible with newer kernel version, up to 4.13.13, and the relative network card drivers. we have now to configure the source code so that it will be built according to our needs (there are several configuration that can be used, we will use the fully-preempible one with dedicated network devices and drivers, see https://www.etherlab.org/en/ethercat/index.php for the details).

- apt-get install dh-autoreconf

- ./bootstrap
- ./configure –disable-8139too –enable-e1000e –enable-generic
- make -j4
- make modules -j4
- make modules_install
- make install

Now the ethercat master is installed in "/opt/etherlab" and has to be configured at application level.

- cd /opt/etherlab/
- mkdir /etc/sysconfig
- cp etc/sysconfig/ethercat /etc/sysconfig/
- ln -s /opt/etherlab/etc/init.d/ethercat /etc/init.d/ethercat
- /usr/lib/insserv/insserv /etc/init.d/ethercat
- lshw -class network (and take note of the MAC ADDRESS (00:1b:21:22:04:61 here) and DRIVER (e1000e here) of the network card you will be use to comunicate with the ethercat drivers)
- gedit /etc/sysconfig/ethercat and edit:

  ```
  MASTER0_DEVICE="00:1b:21:22:04:61"
  DEVICE_MODULES="e1000e"
  ```

Now we are good to go and we can test if the master can work properly with:

```
root@labpc-desktop:/opt/etherlab# ./etc/init.d/ethercat start
Starting EtherCAT master 1.5.2  done
```

At this point, if you reboot the pc, at startup of the realtime kernel the ethercat master will be automatically loaded, so that it can be used right away by any application. If for any reason you are not using e1000e as the ethercat communication port, but the r8169 one, you might experience the failure of the driver to load at boot. This is still an unresolved issue by the developers but a pretty easy workaround is to shut down the pc and unplug it from its power cord: in this way the lan card is initialized again and will work properly next time you boot with the real time kernel.

Chapter 3

An application Example: Running an EasyCAT

When developing your ethercat master for a *rt* linux OS, it is useful to have a straightforward test bench that will help you gain the bare minimum knowledge of the system you are using. Etherlab library it is quite simple to use, if you read the documentation properly, but still it is not banal. When interfacing the master with industrial equipment (i.e. Beckhoff hardware) every sort of error may arise and it is fundamental distinguish between master generated (you committed a mistake in the use of etherlab library), hardware generated (something is wrong with your equipment, check the wiring and the power source) and user generated (something is wrong with your code - this is the most frequent, of course, even when you become proficient with ethercat master).

Easycat Arduino Shield is a (relatively) economical tool that can be bought at http://www.bausano.net/en/hardware/ethercat-e-arduino/easycat.html for around 50 €. In addition, you will need any arduino board among Uno, Mega or Due. For this example a 32-bit Arduino Due will be used (https://store.arduino.cc/arduino-due) at a price of 34 € for the original. With less than 100 € you can build a fully programmable ethercat slave that can be proficiently used as an input/output manager, additional processing unit, actuator controller, etc.. The downside (if you feel it that way) is that you have to program to do exactly what you need. As a matter of fact, an arduino due with an ethernet2 shield can be used even as a master ethercat, with the library provided by EasyCat developers.

3.1 Easycat Overview

The Shield EasyCAT allows to exchange on the bus EtherCAT® 32 byte in input and 32 byte in output, configurable up to 128 byte. The communication is totally managed in Hardware and the exchange of data with the sketch Arduino is made through a library furnished with the EasyCAT together with the file XML EtherCAT® Configuration File (ESI). The EasyCAT Shield can support three types of synchronization: Free RUN, SM Sync and Distributed Clocks. In addition, the XML file can be edited in the configurable version so that the user can customize the input/output PDO to match its needs, as long as their total size does not exceeds the 254 (128+128) bytes limit.

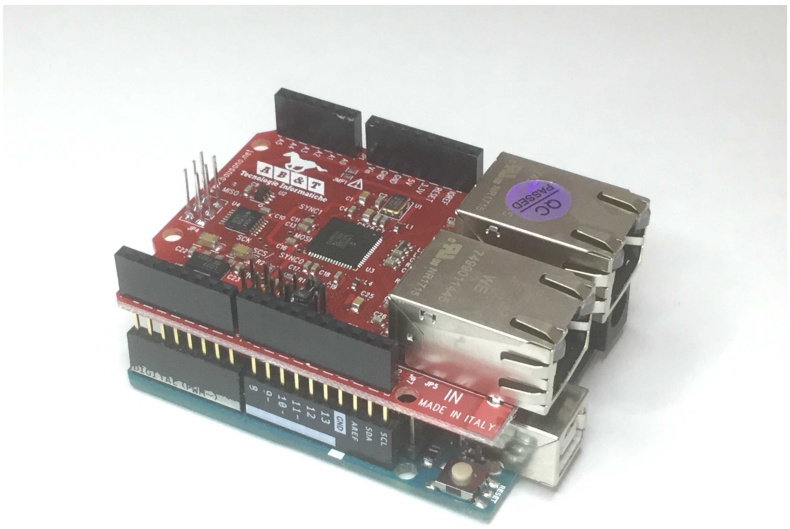

Figure 3.1: Arduino Uno with EasyCAT Shield

At http://www.bausano.net/en/hardware/ethercat-e-arduino/easycat.html several files can be downloaded:

- Easycat Library, an arduino library necessary to program the board as an ethercat slave;
- an XML configuration File;
- Easycat EEPROM programmer tools and Easycat EEPROM BIN files, which are tools that allow the user to reconfigure the number and type of PDOs of the ethercat Slave, within a windows environment (the shield is shipped with a 32+32 bytes, single byte input/output PDOs. Most of the time you'll have to use this tool in order to reprogram the PDOs according to your needs)
- EasyMASTER Library, a file which contains several tools useful to program the arduino board as an ethercat master (yet untested feature by the author of these notes). Beware that if you change the default configuration, you'll need to edit some definition inside the arduino library header EasyCAT.h (only for advanced user)

Fundamentally, in order to verify that the ethercat master library you installed according to the previous chapter is working and to gain a basic knowledge of the former, you can stick with the default 32+32 bytes Input/Output PDOs feature and begin programming your board.

3.2 Programming the Arduino Board + Easycat Shield as an Ethercat Slave

The best way to start developing your code for the Ethercat Slave is to load one of the built in examples and use it as a programming example. Once you have installed at least the Easycat Library, you should be able to load TestEasyCAT.ino.

There are several fundamental parts in this sketch you need to take note:

- #include ”EasyCAT.h” - because it’s the main EasyCAT library;
- #include <SPI.h> - because the EasyCAT shield uses the SPI protocol to communicate with the arduino board processor;
- EasyCAT EASYCAT - declaration of the easycat object;
- EASYCAT.Init() - method to call inside the loop function, to initialize the Ethercat shield;
- EASYCAT.MainTask(); - method to call inside loop to retrieve the data from the ethercat shield (it can be used without the delay to avoid lag in the communication with the master)
- Application(); - this method can be actually called whenever you see fit and can have the name you want: it is the code that will be executed cyclically, at the maximum speed allowed by the cpu of your board and the length of your code
- EASYCAT.BufferOut.Byte[*] it is the output buffer of the MASTER so it contains information provided by the pc master ethercat. Its information must be read in the arduino code.
- EASYCAT.BufferIn.Byte[*] it is the input buffer of the MASTER so it contains information provided by the arduino EasyCat Ethercat slave. Its information must be written in the arduino code.

In this example, we will program the Application() function so as to change the luminosity of the onboard led according to the value of the PDOs, using the byte value as the pwm command value for the led.

For simplicity sake, the arduino code is simply given here, with some remarks:

- Every loop of the application 3 bytes of information are read from the output buffer of the master: the commanded status of the slave (working or idle), the commanded slope of the led brightness (1 rising, 0 falling) for handshaking purposes and the commanded led pwm frequency (aka brightness).
- Some cheks are performed in order to implement the handshaking and if successful the frequency is changed accordingly and the handshaking output is written in the input buffer of the master.

```
#include "EasyCAT.h"
#include <SPI.h>

EasyCAT EASYCAT;

const int onboardLed = 13;
byte slaveStatus = 0; //ControlWord of out slave:
//0 means application not running, 1 means application running
byte slope = 0; //Handshaking flag: it is the internal status
```

```
//of the led blinking  slope. if 1, the brightness is increasing,
//if 0 decreasing.
byte frequency = 0; // The commanded pwm frequency from the master

void setup() {
pinMode (onboardLed,OUTPUT);
analogWrite(onboardLed,frequency);
Serial.begin(115200);
if (EASYCAT.Init() == true)
{
Serial.print ("initialized");
}
else
{
Serial.print ("initialization failed");
pinMode(13, OUTPUT);
while(1)
{
digitalWrite (13, LOW);
delay(500);
digitalWrite (13, HIGH);
delay(500);
}
}
}

void loop() {
EASYCAT.MainTask();
Application();
}

void Application() {
slaveStatus = EASYCAT.BufferOut.Byte[0];
if (slaveStatus) {
byte tempSlope =  EASYCAT.BufferOut.Byte[1];
byte tempFrequency = EASYCAT.BufferOut.Byte[2];

if (tempFrequency-frequency>0) {
slope = 1;
} else {
slope = 0;
}
byte tempFlag;
if (tempSlope==slope) {
tempFlag = 1; //Handshaking Successful
frequency = tempFrequency;
analogWrite(onboardLed,frequency);
}
```

```
else {
tempFlag = 0; //An error Occurred, Master and Slave not in sync
analogWrite(onboardLed,0);
}
EASYCAT.BufferIn.Byte[0] = tempFlag;
}
}
```

Chapter 4

Grab CableRobot

Chapter 5

Conclusion