# ANEXO 1

## Instalación, configuración y compilación de Xenomai, RTNet y LinuxCNC:

### Procedimiento general:

Como guía inicial se usó el siguiente artículo del wiki de linuxcnc.org, escrito por Andy Pug:

<http://wiki.linuxcnc.org/cgi-bin/wiki.pl?Mesa7i80_Driver_For_Linuxcnc_On_Xenomai>

#### Pre-chequeos que hay que realizar:

1. Instalar Python 2.6
2. Agregar al root en el grupo de usuario xenomai:
   1. $sudo adduser root xenomai
   2. logout y login

#### Instalación de linuxCNC real time:

Al momento de configurar la instalación del LinuxCNC, para no generar errores es necesario quitar la librería MODBUS y pasar los scripts para la configuración de Tlc y Tk, el comando sería el siguiente:

./configure --with-tclConfig=/usr/lib/tcl8.5/tclConfig.sh

--with-tkConfig=/usr/lib/tk8.5/tkConfig.sh

--without-libmodbus

#### Trouble shooting de instalación

Estos links pueden ayudar con problemas durante la instalación de linuxcnc sobre xenomai:

<http://wiki.linuxcnc.org/cgi-bin/wiki.pl?XenomaiKernelPackages>

<http://wiki.linuxcnc.org/cgi-bin/wiki.pl?XenomaiKernel>

<http://wiki.linuxcnc.org/cgi-bin/wiki.pl?NewRTInstall>

Para el toubleshooting de Xenomai, ver los siguientes links:

<http://www.berabera.info/oldblog/lenglet/howtos/realtimelinuxhowto/>

<http://wiki.linuxcnc.org/cgi-bin/wiki.pl?XenomaiKernelPackages>

<http://www.xenomai.org/documentation/xenomai-2.6/html/TROUBLESHOOTING/>

<http://www.xenomai.org/documentation/xenomai-2.6/html/xeno-test/index.html>

# 

# ANEXO 2:

## RTNET

### Instalación

Previo a la instalación del RTnet, se debe instalar el Xenomai (seguir pasos en <http://wiki.linuxcnc.org/cgi-bin/wiki.pl?XenomaiKernelPackages>).

A la fecha 28/02/2014 para poder compilar RTnet son necesarios los kernel headers. Para instalarlos dar el siguiente comando:

$ sudo apt-get install linux-headers-3.5.7-xenomai-2.6.2.1

Luego es necesario crear un link simbólico en el directorio que se va a realizar el “build”

$ sudo ln -s /usr/src/linux-headers-3.5.7-xenomai-2.6.2.1/ /lib/modules/3.5.7-xenomai-2.6.2.1/build

Para instalar el RTnet se debe bajar el paquete desde el github:

$ git clone git://rtnet.git.sourceforge.net/gitroot/rtnet/rtnet rtnet-0.9.13  
$ cd rtnet-0.9.13

### 

### 

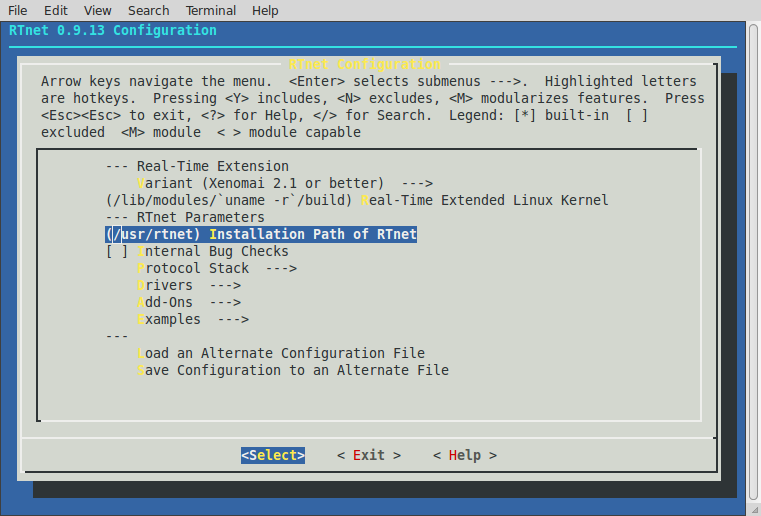
### Configuración sin RTcfg y conRTcap.

La configuración se hace usando la herramienta de configuración menuconfig:

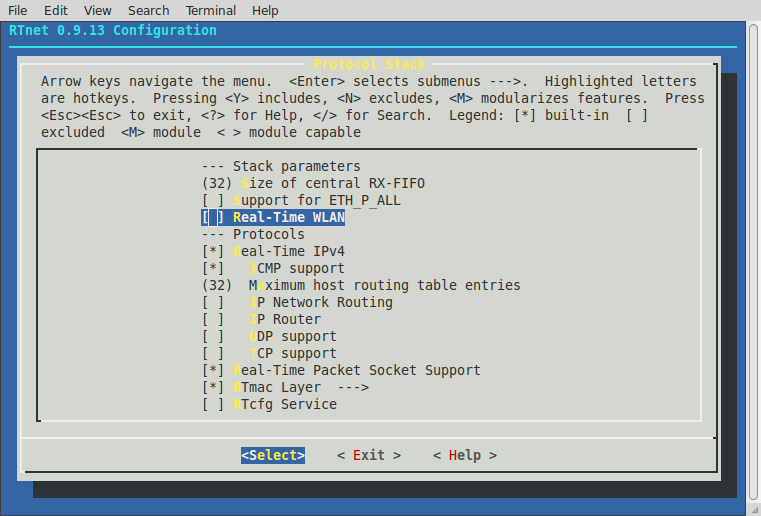
$ make menuconfig

Seleccionar las siguientes opciones:

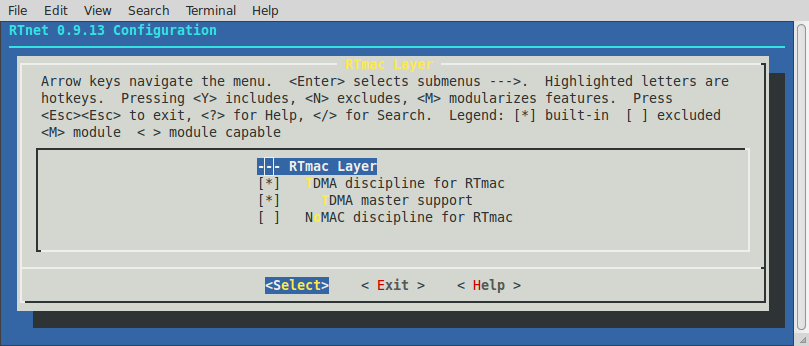
--- Real-Time Extension ---> Variant (Xenomai 2.1 or better)  
 --- RTnet Parameters ---> (/usr/rtnet) Installation Path of RTnet



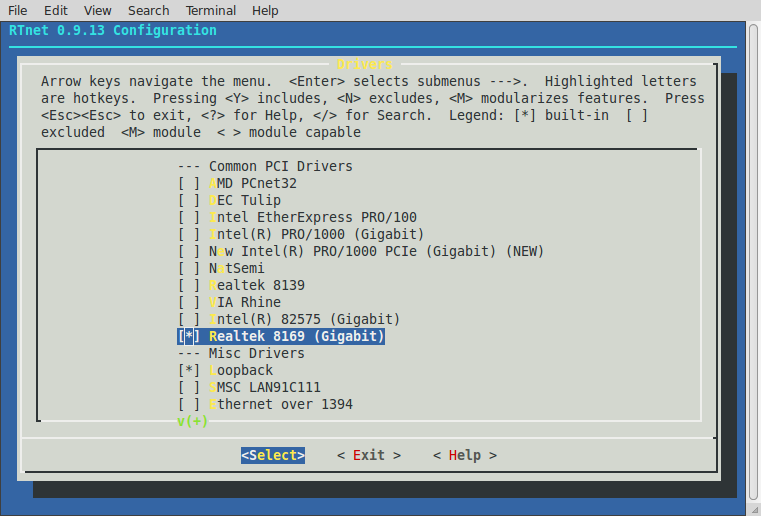
Protocol Stack --->



Se quita el RTcfg Service y el RTmac Layer se configura como sigue:

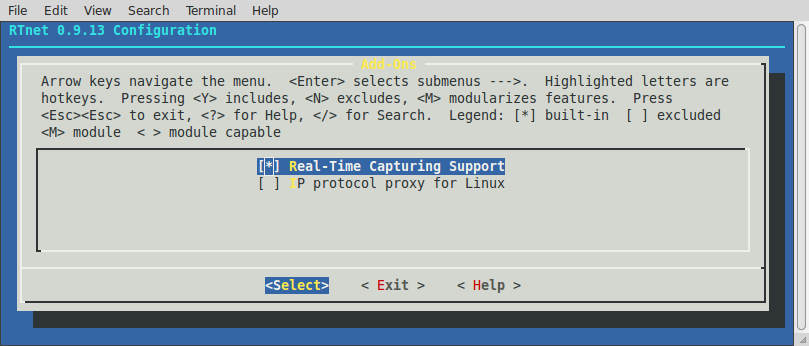


Drivers



Se selecciona el driver de la NIC usada y el loopback.

En Add-Ons se configura RTcap para habilitar la captura con wireshark:



**Compilación:**

Se ejecutan los comandos para la compilación:

$ make  
$ sudo make install

### Archivos de configuración rtnet.conf y tdma.conf.

El archivo principal de configuración es el rtnet.conf, en este se debe definir:

…

RT\_DRIVER="rt\_r8169"

…

IPADDR="192.168.0.1" //Poner la ip que con la que se quiere configurar el maestro

NETMASK="255.255.255.0"

…

RTCAP="yes" //Este flag habilita del RTcap para poder usar el wireshark

…

TDMA\_MODE="master"

…

TDMA\_SLAVES="192.168.0.2" // Dirección ip de o los escalvos

…

TDMA\_CYCLE="1000" //ciclo de trabajo del TDM

TDMA\_OFFSET="50"//duración del time slot

En caso de querer definir los esclavos a nivel de dirección MAC se debe habilitar usar el archivo tdma.conf en el rtnet.conf.

…

TDMA\_CONFIG="${prefix}/etc/tdma.conf"

En el archivo tdma.conf, entonces se definirán los parámetros que “sobrescriben” a los definidos en el rtnet.con:

master:

ip 192.168.3.1

mac F8:1A:67:03:E4:8F

cycle 1000

slot 0 50

backup-master: //no fue usado

slave:

ip 192.168.3.2

mac F8:1A:67:03:EB:84

slot 0 100

slave:

ip 192.168.3.3

mac F8:1A:67:03:EB:85

slot 0 150

**Corriendo RTnet, levantando la interfaz rteth:**

Primero remover el driver no-realtime de la tarjeta de red (ej. Realtek 8169):

$sudo rmmod r8169

$sudo /usr/rtnet/sbin/rtnet start &

Nota: Puede dar un mensaje que no encuentra el RTcfg pero no es un error. Está OK sí dice “Waiting for slaves…”. Verificar que las interfaces estén arriba con “ifconfig -a” y listar los módulos real time en kernel con lsmod para ver que estén cargados.

# 

# 

# 

# ANEXO 3:

## Canfestival

### Compilacion e instalacion

Obtener el paquete de instalación de Canfestival en <http://www.canfestival.org/code> (el lugar de descarga al 28/02/2014 es dev.automforge.net/CanFestival-3/archive/a82d867e7850.tar.gz).

Luego de descargar del paquete, se debe copiar en la carpeta .../Canfestival/drivers/, la carpeta can\_rtnet provista en el CD. Antes de compilar ver “Instalación del can\_rtnet”. La compilación de la librería con el driver can\_rtnet son tres pasos:

Configurar con Xenomai (timers=xeno) y driver can\_rtnet (can=rtnet):

$configure --timers=xeno --can=rtnet

Hacer el make y make install:

$make

$sudo make install

### Desarrollo del driver can\_rtnet

Dijimos que CanFestival tiene previsto incorporar drivers desarrollados para la interfaz que se desea. En nuestro caso fue necesario crear un driver que use el manejo de sockets y envío/recepción de datos según el API de RTnet, pero transportando tramas CANOpen como payload.

Para esto se definió un tipo de trama Ethernet y para los campos de CANOpen se usaron las estructuras de datos definidas en rtcan.h (ver http://www.xenomai.org/documentation/trunk/html/api/rtcan\_8h.html).

#### Definicion del Ethertype 0x9023 canrtnet.

El Ethertype 0x9023 fue definido como canrtnet, para el transporte de tramas CANOpen sobre RTnet. Entonces, la trama será como en la siguente

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Preámbulo** | **Dirección Destino** | **Dirección Origen** | **Tipo** | **Datos** | **CRC** |
| 8 Bytes | 6 Bytes | 6 Bytes | 0x9023 (2 Bytes) | **46 Bytes** | 4 Bytes |

En el campo “Datos” se ubica la trama can definida por rtcan.h, y tiene la siguiente forma:

|  |  |  |
| --- | --- | --- |
| **COBID (can\_id)** | **SIZE (can\_dlc)** | **Datos** |
| **4 Bytes** | **1 Byte** | **1 a 41 Bytes** |

**Instalación del can\_rtnet**

Tener en cuenta que la alineación de campo data[] del can\_frame de rtcan.h define el campo con alineación a 8 bytes, esto agrega 3 bytes que no se utilizan en la trama, eliminando el atributo de alineación en el archivo rtcan.h se evita esto:

Cambiar linea:

uint8\_t data[8] \_\_attribute\_\_ ((aligned(8)));

por:

uint8\_t data[8];

ANEXO 4:

## LinuxCNC:

### HAL y sus componentes.

El Hardware Abstraction Layer del LinuxCNC, nos permite programar en lenguaje ANSI C o Python, un componente de Software que emula el un componente de hardware.

Así el componente tendrá entradas y/o salidas que serán posible conectar a con las de otros componentes.

La sintaxis del componente tiene ciertas sentencias que son interpretadas por un “compilador” que las interpreta y genera los objetos estáticos “.o” y los objetos compartidos “.so” (que pueden ser vinculados dinámicamente a un programa) para ser usados dentro del HAL. Nosotros usaremos principalmente el shell del HAL, el *halcmd*, aunque está incorporado a la GUI del LinuxCNC.

Para usar el compilador de componentes hay que primero correr el script rip-environment, que nos permite ejecutarlo. El comando del compilador es “comp” y hace uso del gcc con determinados argumentos.

Nota: Debe estar cargado el run in place environment (rip-environment) de lo contrario el comando comp que se usará será el del programa compare que es default de GNU/Linux y sistemas Unix en general.

Para tener un comprensión del HAL referirse al HAL Manual que es parte de la documentación de LinuxCNC.

### Compilación de componentes con comp y “manual”.

La posibilidades del comando comp son:

1. $comp micomponente.comp // genera el “.c” micomponente.c
2. $comp --compile micomponente.comp // genera el “.o” micomponente.o
3. $comp --install micomponente.com // genera el “.o” y “.so”, compiando a micomponente.so a ../linuxcnc/rtlib/ para poder usarlo desde el halcmd.

Ahora para el caso de la librería CanFestival, es necesario vincular tanto las cabeceras “.h” como los objetos. En el caso del comp, no es posible pasarle argumentos distintos, así que para un “.comp” que use la librería CanFestival, los pasos para su compilación son:

1. Cargar las variables de entorno:
   1. LD\_LIBRARY\_PATH con los path de las lib necesarias
      1. $export LD\_LIBRARY\_PATH=/usr/local/lib:/home/cnc/linuxcnc-rtnet/lib:/usr/realtime-2.6.32-122-rtai/lib:/usr/lib:/lib
   2. C\_INCLUDE\_PATH con los path de las cabeceras necesarias
      1. $export C\_INCLUDE\_PATH=/usr/rtnet/include:/usr/include/xenomai:/usr/include:/usr/local/include/canfestival:/home/cnc/CanFestival/include:/home/cnc/CanFestival/include/unix:/home/cnc/CanFestival/driver/can\_rtnet:/home/cnc/CanFestival/include/timers\_xeno:/home/cnc/linuxcnc-rtnet/include:/home/cnc/linuxcnc-rtnet/src/hal/components
2. Generar el archivo “.c”:
   1. $comp micomponente.comp
3. Compilar para generar el objeto binario “.o”:
   1. $gcc -O2 -g -I. -I/include -I/home/cnc/CanFestival/include -I/home/cnc/CanFestival/include/unix -I/home/cnc/CanFestival/include/can\_rtnet -I/home/cnc/CanFestival/include/timers\_xeno -DRTAPI -D\_GNU\_SOURCE -Drealtime -D\_\_MODULE\_\_ -I/home/cnc/linuxcnc-rtnet/include -Wframe-larger-than=2560 -fPIC -I/home/cnc/linuxcnc-rtnet/src/hal/components -I/home/cnc/linuxcnc-rtnet/src/hal/components -o micompoenente.o -c micompomente.c
4. Vincular el “.o” con las librerías CanFestival:
   1. $gcc -shared micomponente.o miDiccionarioObjetos.o -o micomponente.so -Wl,-whole-archive -lcanfestival -lcanfestival\_unix -lnative -lxenomai -lpthread -lrt -lrtdm -ldl -Wl,-no-whole-archive
5. Se copia el archivo .so al directorio rtlib:
   1. $cp micomponente.so /home/cnc/linuxcnc-rtnet/rtlib/

### Configuración del canopen.comp desde el halcmd:

Un script para cargar el canopen.comp en desde el halcmd se muestra como ejemplo. Modificar sus parámetros según el uso:

halcmd: loadrt canopen inputs=24831 outputs=24675,25600 slavenodeid=2 masternodeid=1

halcmd: loadrt threads name1=servo-thread period1=10000000

halcmd: addf canopen.0.update servo-thread

halcmd: start

halcmd: show

ANEXO 5:

## Hardware del Esclavo CANopen/RTnet:

### Placa de desarrollo XMOS

Información adjunta en archivos:

Anexo/Hardware\_Esclavo/XMOS-SliceKit-Hardware-Specification.pdf

Anexo/Hardware\_Esclavo/XMOS-XA-SK-GPIO-hwguide.pdf

Anexo/Hardware\_Esclavo/XMOS-XA-SK-ETH100-hwguide.pdf

Anexo/Hardware\_Esclavo/Esquematico\_Placa\_Madre\_XMOS.pdf

Anexo/Hardware\_Esclavo/Esquematico\_XMOS\_GPIO.pdf

### Módulo inalámbrico

Información adjunta en archivos:

Anexo/Hardware\_Esclavo/Hoja\_datos\_nRF24L01P.pdf

Anexo/Hardware\_Esclavo/Conexiones\_modulo\_nRF24L01P.pdf

ANEXO 6:

## Hardware del Pendulo de Furuta:

### Información Mecánica/Electromecánica

Se adjunta en archivos:

Anexo/Hardware\_Furuta/Ensamble\_Mecanico\_Furuta.pdf

Anexo/Hardware\_Furuta/Piezas\_Mecanicas\_Furuta.pdf

Anexo/Hardware\_Furuta/Servo\_Linix\_85W-1.jpg

Anexo/Hardware\_Furuta/Servo\_Linix\_85W-2.jpg

Anexo/Hardware\_Furuta/Servo\_Linix\_85W-3.jpg

### 

### Información Eléctrica

Se adjunta en archivos:

Anexo/Hardware\_Furuta/Circuito\_electrico\_Furuta.pdf

Anexo/Hardware\_Furuta/Interfaz\_Inalambrica\_Encoder.pdf

Anexo/Hardware\_Furuta/Impreso\_Interfaz\_Inalambrica.pdf

Anexo/Hardware\_Furuta/Codificadores\_Rotativos\_HEDS-9140.pdf

ANEXO 7:

## Fuentes de Código:

### Driver RTnet para CANfestival

Se adjuntan sus archivos en: /Anexo/Fuentes\_Driver\_RTnet\_CANfestival

### Componente HAL para LinuxCNC

Se adjuntan sus archivos en: /Anexo/Fuentes\_Componente\_HAL\_LinuxCNC

### Configuración de RTnet

Se adjuntan sus archivos en: /Anexo/Configuracion\_RTnet

### Implementación Esclavo CANopen/RTnet sobre XMOS

Se adjuntan sus archivos en: /Anexo/Fuentes\_Esclavo\_XMOS

### Interfaz Inalámbrica para Codificador Rotativo

Se adjuntan sus archivos en: /Anexo/Fuentes\_Interfaz\_Inalambrico