SISTEMAS EMBEBIDOS
DESARROLLO DEL PROYECTO

Sebastián Corzo Gómez – scorzog@unal.edu.co
José Logreira Ávila – jalogreiraa@unal.edu.co

Stamp-Mini

Se eligió utilizar la tarjeta *mini-stamp, d*iseñada por el ingeniero Camargo, que es una versión compacta de una tarjeta con mayores prestaciones. El diseño del PCB fue hecho en Kicad, y la lista de materiales fue elaborada por Camargo, incluyendo los códigos de las partes en Digikey. Los materiales fueron pedidos por Digikey.com, y las PCBs fueron elaboradas en PCBcart en China.

Ensamble de la tarjeta:

Materiales y herramientas necesarias para el ensamble:

- Estación de soldadura: Pinzas de soldadura, Cautín con punta fina.
- Líquido Flux, para limpieza de superficies
- Alcohol isopropílico

Se inicia soldando los componentes pasivos, como los condensadores y las resistencias. Se limpia la PCB con alcohol isopropílico usando un cepillo. Una vez seca se deben estañar todos los pads de los componentes a soldar. Los circuitos integrados no se les debe aplicar mucho estaño a los pads, puesto que al poner el chip, no sería fácil posicionarlo pues los pads estarían abultados.

Se aplica un poco de Flux en los pads del componente a soldar. Con las pinzas para soldadura se coge el componente a soldar por las puntas metalizadas, y se ubica encima de los pads donde este debe ser soldado. El calor derretirá el estaño ya aplicado en los pads, y también quedará adherido a los extremos metalizados del componente. Cuando el componente haya entrado en contacto con el estaño de los pads en sus dos extremos, la pinza se puede retirar, y se verifica si el componente quedó fijado firmemente, o si está levantado. Usando las mismas pinzas de soldadura se pueden realizar reacomodaciones del componente. Es importante siempre usar Flux cuando se solde cada componente, pues limpia la superficie y ayuda a una mejor transferencia de calor.

Al soldar los chips (procesadores y memorias), se debe primero posicionar el IC tal que todas las patas queden alineadas con las pistas a las que van soldadas. Manteniendo fijo el chip, con el cautín de punta fina, se solda solo la pata de un extremo, y la pata del extremo opuesto, pasando ligeramente el cautín sobre la pata a soldar. Solo con estas dos patas soldadas, se verifica que todas las demás patas del IC continúen alineadas con las pistas del PCB. Si no lo están, pasando repetidamente el cautín por las patas soldadas se puede derretir el estaño y realinear el chip completo. Estando satisfechos con la alineación, se procede a soldar el resto de patas del IC, pasando rápidamente el cautín por cada pata individualmente, procurando que no haya estaño aderido a la punta del cautín.

Finalmente, se soldan las regletas laterales de la tarjeta. Cuando la tarjeta esté lista, se lava completamente con alcohol isopropílico, cepillandola con un cepillo de dientes. Se puede verificar con un multímetro que no haya continuidad de las lineas de alimentación y tierra, ni entre los pines de los chips. Esto teniendo en cuenta el esquemático del PCB.

Comunicación con el exterior:

La tarjeta posee múltiples pines de comunicación. Los que se han de usar para la programación y la interfaz con el usuario son:

- USB DP (+) y DM(-): para realizar carga de programas pequeños a la memoria interna del procesador y probar el funcionamiento de la tarjeta.
- Serial Port Rx y Tx: Para la interacción entre el usuario y la tarjeta usando Minicom.

Para la comunicación usando los terminales USB, se usa un cable USB a miniUSB, al cual se le recorta el extremo del terminal miniUSB; a los 4 cables (+5V, GND, DP, DM), se les adapta una regleta hembra tal que se puedan conectar de a pares a la regleta macho de la tarjeta (+5V y GND juntos, y DP y DM juntos). Los terminales +5V y GND energizan toda la tarjeta.

Para la comunicación serial se requiere un IC que sirva de interfaz entre el protocolo USB y serial, pero usando niveles lógicos de 0V y 3.3V que son los que usa el procesador. Esta característica en el mercado se conoce como *"USB to serial - TTL compatible".* En el mercado se consigue un pequeño PCB que tiene montado el conector USB, el IC PL2303hx, y 5 pines de salida, que son Rx, Tx, GND, +5V, +3.3V. Para comunicar la tarjeta con esta interfaz se usa un cable ribbon de 5 hilos, y en ambos extremos se le adapta regletas hembra para poderse conectar a los pines Tx Rx del PL2303hx, y a los pines Rx y Tx de la tarjeta.

Prueba de funcionamiento:

Las instrucciones a continuación fueron entregadas por el Ing. Camargo para la prueba de la Mini-stamp. Se requiere de una memoria micro SD de 2GB o más, y el cable USB previamente fabricado. Previamente se requiere también instalar *libusb,* que se encuentra disponible en el gestor de software de Ubuntu, aunque la opción que nos funcionó correctamente fue descargar e instalar manualmente *libusb* desde el sitio oficial *http://www.libusb.org/.*

Descarga de Lua:
$ wget http://www.lua.org/ftp/lua-5.2.2.tar.gz
$ tar zxvf lua-5.2.2.tar.gz
$ cd lua-5.2.2
$ make linux
$ sudo make install
$ sudo cp /usr/local/lib/liblua.a /usr/lib/

Lua es un lenguaje compacto, interpretado, ligero y sencillo, y es uno de los que soporta el i.MX .

Crear las particiones necesarias en la SD, formatear la memoria SD y grabar el bootloader y el sistema de archivos:

Se necesitan dos particiones en la SD, la primera almacenará el boot-loader y la imágen del kernel; la segunda almacenará el sistema de archivos. Para los siguientes comandos se asume que la tarjeta es detectada como /dev/sdb (para conocer el dispositivo asignado a la tarjeta por Linux se debe insertar la tarjeta y ejecutar el comando $ dmesg)

Desmontar la partición y ejecutar fdisk:
$ sudo umount /dev/sdb*
$ sudo fdisk /dev/sdb

Borrar las particiones existentes: Dentro de fdisk se debe escribir "d" (delete a partition)

```
Command (m for help): d
Partition number (1-4):
```

En el campo "Partition number" se debe escribir el número de la partición a ser eliminada, el proceso se debe repetir hasta borrar todas las particiones existentes.

Crear una partición de 200MB : Dentro de fdisk se debe escribir "n" (add a new partition) . Seleccionar partición primaria: Escribir "p"

```
Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
```

Fijar el comienzo de la partición. Al mensaje:

```
First sector (2048-7626751, default 2048):
```

Se debe oprimir la tecla enter para asignar el valor por defecto. A lo que la aplicación responde:

```
Using default value 2048
```

Fijar el final de la partición. Al mensaje:

```
Last sector, +sectors or +size{K,M,G} (2048-7626751, default 7626751):
```

Ingresar +200M A lo que el sistema responde:

```
Partition 1 of type Linux and of size 200 MiB is set
```

Crear la segunda partición en el espacio sobrante en la SD : Dentro de fdisk se debe escribir "n" (add a new partition) . Seleccionar partición primaria: Escribir "p" . Asignar el número de la partición a "2"

```
Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1-4, default 2): 2
```

Fijar el comienzo de la partición: Dejar el valor por defecto
Fijar el final de la partición: Dejar el valor por defecto

```
First sector (411648-7626751, default 411648):
Using default value 411648
Last sector, +sectors or +size{K,M,G} (411648-7626751, default 7626751):
Using default value 7626751
Partition 2 of type Linux and of size 3.5 GiB is set
```

Cambiar el tipo de la partición 1 a OnTrack DM6 (53) : Dentro de fdisk escribir "t" (change a partition's system id) . Escribir "53" . Este es el formato de partición que Freescale diseñó para el i.MX.

```
Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): 53
Changed system type of partition 1 to 53 (OnTrack DM6 Aux3)
```

Cambiar el tipo de la partición 2 a Linux (83) : Dentro de fdisk escribir "t" (change a partition's system id) . Escribir "83"

```
Command (m for help): t
Partition number (1-4): 2
Hex code (type L to list codes): 83
```

Al finalizar se debe tener la siguiente tabla de particiones, la cual se obtiene al ejecutar el comando "p" (print the partition table);

```
   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            2048      411647      204800   53  OnTrack DM6 Aux3
/dev/sdb2          411648     7626751     3607552   83  Linux
```

Una vez finalizada la creación de las particiones se deben guardar los cambios en la tabla de particiones de la SD, para esto debe ejecutar el comando "w" (write table to disk and exit)

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

Formatear la segunda partición como ext3:

```
$    sudo mkfs.ext3 /dev/sdb2
```

Almacenar el boot-loader y la imágen del kernel a la SD: El bootloader y la imágen se encuentran en la carpeta Factory_Test/basic/files.

```
$    sudo dd if=sd_mmc_bootstream.raw of=/dev/sdb1
$    sudo umount /dev/sdb*
```

Grabar el sistema de archivos a la segunda partición: El .tar.bz2 se encuentra en la carpeta *files.*

```
$    sudo  tar -jxvf buildroot_fs.tar.bz2  -C /media/disk/
```

Cambiar los OTPs para permitir la inicialización por la SD (se realiza solo una vez): Por defecto el iMX233 no permite inicializar desde la SD, para habilitar esta opción es necesario cambiar el valor del registro interno OCOTP.ROM0, esto se realiza una única vez.

Compilación de las herramientas necesarias para la modificación del registro OCOTP.ROM0
Descomprimir y compilar el código fuente: en la carpeta *basic:*

```
$    cd Factory_Test
$    ./build
```

Al finalizar el proceso sin errores deben existir dos nuevos ejecutables en /usr/local/bin:
hwstub_shell sbloader

Cargar la aplicación blink: (alimentar y conectar el cable USB a la stamp): Para asegurarse que la tarjeta es reconocida al conectarla, se ejecuta el comando $ lsusb.

```
load_blink
```

El LED D3 (esquina superior derecha) parpadeará, lo que indica que la placa se encuentra bien soldada y está funcionando de forma adecuada .

Para programar los registros que permiten arrancar desde la SD debe ejecutar el script

```
burn_OTP
```

Y con esto ya puede arrancar con la SD que creó anteriormente.

Se conecta el dispositivo USB a serial, usando el cable ribbon, a los pines Rx y Tx de la tarjeta, alimentandola también con los pines de +5V y GND. Se abre una consola de Minicom donde abre las opciones de configuración del puerto:

$ sudo minicom -s

En la pestaña *serial port setup* se configura el puerto del dispositivo a usar (en nuestro caso: /dev/ttyUSB0), se establece la rata de baudios y los bits de paridad y de inicio como 115200 8N1. No se habilita el flujo de control de hardware ni de software. Se guardan los cambios y se entra en la consola del puerto. Apenas se energiza la tarjeta, en la consola se muestran los comandos iniciales del boot del procesador, y comienza la carga del sistema operativo. Cuando el sistema termina de cargar, el SoC debe pedir el login: root, y el password: root. Al llegar a este paso, todo el sistema cargó correctamente y ya se puede empezar a hacer uso del OS cargado en la tarjeta.

NOTA: El bootloader, el kernel y el sistema de archivos cargados hasta ahora son todos imágenes precompiladas por el Ing. Camargo, que se usan únicamente para probar el funcionamiento inicial de la tarjeta. A continuación se describe cómo compilar desde ceros la imagen del kernel y el sistema de archivos.

Compilando la imagen del Sistema de archivos

Parte del material del curso es el archivo comprimido *buildroot-2013.08.1*. Este contiene un programa llamado *Buildroot.* Es una herramienta que simplifica y automatiza el proceso de creación de un sistema Linux completo para un sistema embebido, usando la compilación cruzada. Para construir el sistema Linux, Buildroot puede generar un set de herramientas de cross-compilación, un sistema de archivos raíz, una imágen del núcleo Linux y un gestor de arranque (bootloader) para la mini-stamp.

Buildroot se puede usar para cualquiera de las tareas anteriormente descritas, en cualquier combinación que se quiera. Soporta múltiples procesadores y viene con múltiples configuraciones preformadas para tarjetas específicas.

En el directorio raíz *buildroot-2013.08.1* se encuentra un Makefile con el cual se realizan todas las acciones necesarias a usar con Buildroot. Al ejecutar:

$ make menuconfig

Se abre una interfaz gráfica en donde se encuentran jerarquizadas todas las opciones de configuración de Buildroot: se puede elegir que clase de sistema de archivos crear, que versión de kernel, que opciones de boot, que paquetes y aplicaciones se desean incluir en el sistema de archivos, que herramientas de desarrollo se desean incluir, etc. Por defecto ya viene un archivo de configuración cargado con las opciones básicas para realizar aplicaciones sencillas con la mini-stamp. Este archivo se llama *.config* y se encuentra en el directorio raíz de Buildroot. Si se realizan cambios en la interfaz de configuración, y se guardan los cambios, este archivo se modifica.

Por ahora, la configuración que viene por defecto es casi suficiente para nuestro propósito por lo que solo es necesario modificar un par de parámetros: en menú *System Configuratio,* se modifica el System *banner: "Welcome to buildroot: Sebastián y Jose"*, y el parámetro *Port to run a getty (logging prompt) on: "ttyAM0"*, que es el puerto serial del SoC a través del cual se va a transmitir a Minicom la consola de Buildroot.

Correr Buildroot con la configuración por defecto creará un sistema de archivos básico y toda la cadena de herramientas del cross-compilador. No creará un bootloader, ni una imágen del kernel

Linux.

Para correr Buildroot, nos ubicamos en el directorio raíz de buildroot, y se ejecuta $ make. Comienza un proceso de descarga de internet de todos los paquetes necesarios, que los va guardando como comprimidos en la carpeta *buildroot-2013.08.1/dl.* Luego empieza a compilarlos usando el cross-compilador seleccionado para el i.MX. Este es un proceso que puede tardar al rededor de 2 horas.

Al llegar a la compilación del paquete llamado *openocd,* muestra un error por un Warning, y no permite continuar con la compilación. Este error es debido a que en el Makefile de este paquete, los warnings están definidos para que los tome como errores. Por esto, algunos warnings que no afectan el funcionamiento del sistema son tomados como errores. Para solucionarlo, nos dirigimos hasta donde se encuentra el Makefile en ejecución, que se encuentra en la ruta: buildroot-2013.08.1/output/build/openocd-0.7.0/src/target, y se abre el Makefile con gedit. Se busca el flag *-Werror,* que está escrito dos veces en todo el archivo. Se borra y se guardan los cambios.

Nuevamente se regresa al directorio raíz de buildroot y se vuelve a correr $ make. Esta vez la compilación continúa sin errores hasta que acaba todas las tareas. El resultado final es que en el directorio *buildroot-2013.08.1/output/images* se encuentra la imágen del sistema de archivos creada, comprimida. Teniendo esta imágen ya creada, se descomprime en la segunda partición de la SD.

Compilando la imagen del Kernel de Linux:

Parte del material de clase son dos archivos comprimidos llamados:

- *linux-2.6.35.3.tar.bz2:* Contiene todo el código fuente del núcleo Linux. Además, también contiene un programa con una interfaz gráfica basada en la librería *curses* (la misma usada para el programa *buildroot*) donde se encuentran múltiples opciones de configuración del kernel.
- *Imx-bootlets-src-10.05.02.tar.bz2*: Son herrmientas para copiar la imagen de linux (creada en la carpeta anterior) a la SD, junto con un conjunto de configuraciones de registros internos del procesador i.MX, necesarios para que este pueda inicializar reguladores de tensión, memorias externas y buses de comunicación.

Estos se descomprimen en un mismo directorio.

Modificación de la variable PATH:

Un primer paso en modificar la variable de entorno PATH, añadiendole la ruta donde se encuentra el cross-compilador que fue creado usando *buildroot.* Esto se realiza modificando el archivo *.bashrc* que se encuentra en el folder del usuario (/home/jose en nuestro caso):

```
$ cd /home/jose
$ gedit .bashrc
```

Al final del archivo se añade la linea de comando:

```
export PATH=$PATH:/home/jose/Documents/EMBEDDED_SYSTEMS/buildroot-2013.08.1/output/host/ usr/bin/
```

Esto modifica la variable PATH añadiendole una nueva ruta de búsqueda de ejecutables. La ruta especificada es donde se encuentra el cross-compilador.

Creación de la imagen de Linux:

Dentro de la carpeta *linux-2.6.35.3* hay un Makefile que será usado para compilar todo el núcleo Linux, usando el cross-compilador generado en la carpeta *buildroot-2013.08.1*.

Para editar las opciones de configuración del kernel, se abre la interfaz gráfica parecida a la existente en *buildroot-2013.08.1,* la cual permite listar y seleccionar las múltiples configuraciones. Para esto se ejecuta en el directorio raíz de l*inux-2.6.35.3*:

$ make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig

donde *ARCH=arm* especifica que se va a compilar para la arquitectura ARM, y CROSS_COMPILE = arm-linux- especifica el cross-compilador que se va a usar. Finalmente, se especifica la directríz a ejecutar: *menuconfig.*

Al ejecutar este comando, se presenta un error de un enlace roto a un archivo. En la carpeta *buildroot-2013.08.1/output/host/usr/bin/* existen múltiples archivos que son enlaces a otros archivos ubicados en esa misma carpeta. El enlace roto se presenta en este archivo: *arm-linux-ld.real,* que apunta a *arm-buildroot-linux-uclibcgnueabi-ld.real.* Este problema se soluciona creando manualmente el enlace, así:

$ ln -s arm-buildroot-linux-uclibcgnueabi-ld.real arm-linux-ld.real

donde la opción "-s" crea un enlace simbólico entre ambos archivos.

Al volver a correr el comando *make* (…) se abre la ventana de la interfaz gráfica que muestra las opciones de configuración del kernel.

Al correr el siguiente comando, se genera la imágen del kernel de linux:

$ make ARCH=arm CROSS_COMPILE=arm-linux-

Esta ejecución se demora de 10 a 15 minutos. La imagen final del Kernel Linux la crea en la carpeta *linux-2.6.35.3/arch/arm/boot,* y tiene como nombre *zImage.* Éste es el Sistema Operativo propiamente dicho, el cual se debe grabar en la primera partición de la SD, usando las herramientas ubicadas en la carpeta *Imx-bootlets-src-10.05.02.*

Pasar la imagen del Kernel a la SD:

La carpeta *Imx-bootlets-src-10.05.02* con la configuración necesaria para el I.MX es proporcionada por Freescale, y contiene la configuración de boot del procesador, donde inicializa los buses de comunicación con las memorias, los reguladores internos de voltaje y registros de configuración.

En el directorio raíz de la carpeta hay un Makefile, el cual se debe correr para pasar la imagen del Kernel a la tarjeta SD. Se debe asegurar que la tarjeta micro SD sea leída como *"sdb",* puesto que así está definida la ruta en el Makefile. Si se usa un adaptador microSD a USB, el sistema leerá la SD como un dispositivo de nombre *"p",* por lo que se debe modificar esta ruta a */dev/p1.*

Para correr el Makefile, se ejecuta el comando siguiente:

$ make ARCH=mx23 CROSS_COMPILE=arm-linux-

Este comando arroja error en la compilación pues no encuentra el comando *"elftosb2",* el cual es el programa encargado de convertir el formato de archivos *elf* (Executable & Linkable File) al formato *sb,* que es el que reconoce el procesador.

Este programa nos lo envió el Ing. Camargo. Es un archivo binario que se debe copiar en la carpeta /usr/bin, y se le deben cambiar los permisos de ejecución:

$ sudo chmod +x /usr/bin/elftosb2

Así, al volver a correr el Makefile, este realiza todos los procesos sin errores, y la imágen del kernel ya queda pasada sin problemas a la SD.

Para comprobar que todo haya salido bien, se inserta la SD en la mini-stamp, se conecta su puerto serial a minicom y se energiza la tarjeta. Primero debe mostrar información de inicialización de todos los registros internos del IMX, y luego empear a cargar el sistema operativo. El mensaje final que aparece es "Welcome to Buildroo2", junto con una petición de login: user: root, password: qwerty. Al iniciar la sesión, ya podemos navegar entre las carpetas del sistema de archivos y asegurarnos que todas contengan los archivos y carpetas necesarias para el funcionamiento del sistema de archivos.

Realizando pruebas con Drivers:

Los drivers son archivos que contenen el código fuente de la configuración de distintos periférios y demás disposotivos externos. Estos realizan las tareas de comunicación entre el OS y el Hardware que se quiere usar. A continuación se describe el uso de un Driver.

En la carpeta Examples/imx233/drivers/blink (incluida en el material del curso) se encuentra un ejemplo de un driver. Esta carpeta solo contiene 2 archivos: blink.c y Makefile. El driver es el blink.c. En este archivo se encuentran definidos los puertos de I/O que va a usar el driver, la configuración de funciones que va a ejecutar, entre otras cosas. Este driver ejecuta una rutina de encendido y apagado 5 veces de un LED, cada vez que recibe un comando de escritura.

Para compilar el driver se usa el Makefile. Pero se debe asegurar que el path a la carpeta raíz de la versión compilada de linux sea el correcto. Además, se debe asegurar que esté bien escrito los nombres de los compiladores, y el nombre del archivo objeto a compilar. En nuestro caso:

```
CC = arm-linux-gcc
KERNEL_SRC = /home/jose/Documents/EMBEDDED_SYSTEMS/linux-2.6.35.3
CROSS_COMPILE = arm-linux-

obj-m += blinker.o
```

Una vez asegurado esto, desde la carpeta que contiene el Makefile sólo se ejecuta:

```
$ make
```

Este proceso arroja un archivo de extensión .ko, que es el driver compilado para la arquitectura deseada. Este archivo se debe copiar en cualquier parte de la SD (nosotros creamos en el directorio raíz: /drivers), y para ejecutarlo, desde la terminal de la mini-stamp se ejecuta el comando:

```
$ insmod blinker.ko
```

Todos los drivers deben tener una función de inicializar (init) y una función de cerrar (exit), las cuales definen algún tipo de configuración del driver, como los pines de I/O que se quieren usar. Al ejecutar el comando anterior, se inserta el módulo en el kernel de linux, y se llama a la función init. En el archivo /proc/devices se crea un registro de todos los drivers presentes, en donde debe aparecer el actualmente creado, llamado "blink". También le asigna un número de identificación. Esto se visualiza al ejecutar:

```
$ cd /proc
$ more devices
```

El número asignado es el 252 en nuestro caso. Para comenzar a usar el driver, se debe crear manualmente el archivo que va a manejar el puerto, así:

$ mknod /dev/blink c 252 0

La "c" quiere decir que la comunicación del driver es de tipo caracter, es decir, que se envían tramas de 8 bits al ejecutar la función de escritura. Para asegurarse que se haya creado correctamente, se pueden listar los dispositivos actualmente presentes en el SoC:

$ ls /dev

Allí debe aparecer creado el blink.
Para comunicarnos con el periférico, una de sus funciones es escribir. Esto se puede hacer así:

$ echo A> /dev/blink

Al recibir el caracter A, el driver ejecuta la rutina de encendido y apagado del LED.

Programando el ADC del SAM3N:

Para poder usar el SAM3N, este requiere de un sistema operativo que lo controle. Se llama ChibiOS, que es un OS muy liviano y orientado a la programación de aplicaciones en tiempo real. Permite crear hilos de procesos paralelamente, y manejar todos los periféricos del SAM3N.

La aplicación que se desee implementar con el SAM3N se escribe en código C, y usando la cadena de herramientas de compilación para el SAM, se genera un archivo binario ch.bin, que es el que contiene tanto el OS que va a controlar el procesador, como el código de la aplicación que se desea correr.

Para pasar este archivo a la memoria del SAM, primero se copia en alguna parte de la SD, y luego se usa la aplicación “openocd” para enviar vía puerto serial, el ch.bin desde la memoria SD, a través del iMX, hasta la memoria del SAM. La aplicación Openocd fue incluida dentro del sistema de archivos cuando fue creado.

Para el caso del uso del ADC, el ejemplo se encuentra en /ChibiOS_mini/demos/ARMCM3-SAM3N-EK/main.c; en esa carpeta también se encuentra el Makefile, así como todos los demás archivos y librerías provistas por Atmel para usar las múltiples funciones preprogramadas. Para compilar el código del ADC junto con el OS: $ make. Esto crea el ch.bin, que se pasa a la SD, junto con múltiples archivos de configuración del SAM, también presentes en la carpeta del main.c

Desde la terminal de la Mini-stamp, en la ubicación del ch.bin:

$ openocd -f stamp_mini_sam3n.cfg

Para crear el nodo de comunicación entre el SAM y el iMX, que va a ser un puerto serial:

$ mknod /dev/ttySP1 c 242 0

El “c” significa: comunicación tipo char: envío de datos por carácter, no por bloques.
El puerto serial está por defecto configurado para una velocidad de transmisión de 38400 baudios, pero el ejemplo del ADC transmite a 115200 baudios. Para cambiar esto:

$ stty -F /dev/ttySP1 115200

Ahora sí, se puede ver lo que está recibiendo el puerto:

$ cat /dev/ttySP1

El proceso se corta con ctrl + C.

Usando Interrupciones en los Drivers:

Otro ejemplo es dado para comprender el uso de interrupciones en los drivers de dispositivo. En la carpeta Examples/imx233/drivers se encuentra el ejemplo IRQ. En el driver de este ejemplo se encuentra definida la interrupción a implementar, que es detectada cuando en el pin definido hay un cambio de nivel lógico.

La estructura de este ejemplo es: ……

La estructura de la interrupción es: ….

La compilación de este driver es muy parecida a la del ejemplo del blinker: se verifican los paths del Makefile, y los compiladores a usar. Se ejecuta el comando make, y el archivo .ko resultante se pasa a la tarjeta. Desde allí se inserta al kernel de Linux (con insmod), y se crea el nodo de comunicación (mknod). El driver ahora tiene la funcionalidad de producir interrupciones en el sistema.

Montaje de la Red Inlámbrica

COMANDOS:

$ ifconfig: muestra el estado de las interfaces de red actualmente activas
$ iwconfig: similar a ifconfig, pero solo para redes inalámbricas. No sale la misma info
$ ping <IP_dir> realiza test de comunicación

Se eligió usar la tarjeta de red USB Minicasadora de 3bumen, con chip RT3070 de Ralink.

Al conectar la minicazadora, en $ifconfig debería salir la wlan. No hemos hecho la prueba

PROGRAMAS:

Servidor DHCPD: Dynamic Host Configuration Protocol Daemon
DHCPD: DHCO daemon: un daemon es un programa que corre en segundo plano, sin interfz directa con el usuario. DHCPD es el programa que corre en segundo plano en un servidor, que gestiona la asignación de IP dinámicamente.

The DHCP server then offers the "lease" of an IP address to the client, which the client is free to request or ignore. If the client requests it and the server acknowledges it, then the client is permitted to use that IP address for the "lease time" specified by the server. At some point before the lease expires, the client must re-request the same IP address if it wishes to continue to use it. Issued IP addresses are tracked by dhcpd through a record in the dhcpd.leases file

dhcpd.conf: the network administrator allocates address pools in each subnet and enters them into the dhcpd.conf(5) file.

On startup, dhcpd reads the *dhcpd.conf* file and stores a list of available addresses on each subnet in memory. When a client requests an address using the DHCP protocol, dhcpd allocates an address for it. Each client is assigned a lease, which expires after an amount of time chosen by the administrator (by default, one day). Before leases expire, the clients to which leases are assigned are expected to renew them in order to continue to use the addresses. Once a lease has expired, the client to which that lease was assigned is no longer permitted to use the leased IP address.

Whenever changes are made to the dhcpd.conf file, dhcpd must be restarted. To restart dhcpd, send a SIGTERM (signal 15) to the process ID contained in */var/run/dhcpd.pid*, and then re-invoke dhcpd. Because the DHCP server database is not as lightweight as a BOOTP

database, dhcpd does not automatically restart itself when it sees a change to the dhcpd.conf file.

biblio:
http://www.linuxcommand.org/man_pages/dhcpd8.html
http://en.wikipedia.org/wiki/Lease

Hostapd is a user space daemon for wireless access point and authentication servers.
hostapd is configured using a text file that lists all the configuration parameters.
The current version supports Linux (Host AP, madwifi, mac80211-based drivers) and FreeBSD (net80211).
All **new** mac80211 based drivers that implement AP (Access Points) functionality are supported with hostapd's nl80211 driver.
nl80211 is the new 802.11 netlink interface public header. Together with cfg80211 it is intended to replace Wireless-Extensions. nl80211 and cfg80211 are still under development.
You will also want to do a survey of your area to find the channel that has the fewest other APs on it. When choosing which channel to use, it is important to remember that the channels overlap with any channels that are within 20MHz.

Biblio:
http://hostap.epitest.fi/hostapd/
http://wireless.kernel.org/en/users/Documentation/hostapd

COPIAR ARCHIVOS A LA MINI-STAMP POR LA RED:

SCP significa **S**ecure **C**o**P**y es una pata más de SSH, que permite transferir archivos o carpetas entre computadores. La sintaxis es bien simple:

$ scp *archivo* usuario@servidor.com:ruta

Y para copiar a la inversa, desde el computador remoto al tuyo, simplemente tienes que invertir el orden de los elementos:

$ scp *usuario*@*servidor.com*:*ruta/archivo ruta_local*

O sea por ejemplo, si quisiéramos mandar algo al servidor:

$ scp hola.txt tomas@bootlog.cl:/www/sitio

Mandaría el archivo **hola.txt** y lo dejaría en la carpeta **/www/sitio** en el servidor **bootlog.cl**. También puedes mandar carpetas completas (con):

$ scp **-r** viajealsur/ tomas@bootlog.cl:/www/sitio/fotos

Así mandaría la carpeta **/viajealsur** completa a **/www/sitio/fotos**.
Ahora, el mismo proceso a la inversa sería:

$ scp **-r** tomas@bootlog.cl:/www/sitio/fotos/viajealsur ~

Esto copiaría la carpeta **viajealsur/** del servidor a mi carpeta **/home**.

http://pintucoperu.wordpress.com/2007/11/27/como-usar-scp-el-complemento-de-ssh-para-transmitir-archivos-y-carpetas/

En el caso de la ministamp:
$ scp <ruta_al_archivo> root@192.168.50.1:<ruta_de_destino>

AHORA SÍ EL MONTAJE DE LA RED:

Inicialmente se siguieron las instrucciones de los archivos .ps enviados por Camargo: network.ps y rt3070_1.ps

Darle soporte de Drivers al Kernel linux: El chip es de Ralink. Es el rt3070l. Desde la carpeta de linux, se ejecuta la interfaz de menuconfig:

$ make menuconfig
o
$ make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig
-->No sé cual es la diferencia entre los dos

Se le da soporte a los chips de Ralink como dice el apéndice B del documento. Se vuelve a compilar la imagen del kernel, que queda ubicada en <linux>/arch/arm/boot. Usando iMXbootlets se pasa la imágen a la SD.

Desde el sistema de archivos de la SD (mini-stamp), se deben modificar algunos archivos para configurar la red, según la guia networks.ps:

- Configurar dhcpd: en el archivo etc/dhcp/dhcpd.conf , se cambian las direcciones IP según la guia.
- Se crea el archivo que no existía etc/default/isc-dhcp-server , con la linea INTERFACES=”wlan0”
- Se configura hostapd: el archivo /etc/hostapd/hostapd.conf se modifica, se le da nombre a la red ssid=osciloscope. *Algunas de las líneas de opciones de hostapd.conf que sugería la guia se comentaron, según les había funcionado a otro grupo.*
- Configurar red local del servidor: */etc/network/interfaces .* Se debe poner una IP fija para el servidor. *Algunas de las líneas de código que sugiere la guia se cambiaron, según Camargo indicó en archivos pasados por correo.*

Adicionlmente, como soporte para manejar la tarjeta de red, al parecer en la carpeta /lib/firmware no están los archivos necesarios para manejar la tarjeta, por lo que Camargo nos pasó unos archivos de firmware .bin adicionales que se metieron en esta carpeta.

Como a varios no les funcionaba la red, Camargo comparó dos sistemas de archivos: el de la mini-stamp de él, y el de la del grupo de Santiago. Luego de eso, nos envió archivos que estaban distintos para añadir a nuestra SD:

- En init.d: se adicionaron los 2 archivos que envió.
- En Network: se modificó interfaces, cambiando las opciones que decía la guia.
- Se agregó la carpeta etc/wireless, que no existía
- inittab no tuvo cambios
- wpa_suplicant.conf sí cambió: todos los parámetros de Network se agregaron.
- En /lib/firmware se agregaron todos los .bin que envió.
- En /lib se añadieron todos los demás archivos que envió
- en /root se metieron los archivos y carpetas mandados

Adicionalmente, Santiago envió unos pasos para conectarse de manera manual a la tarjeta, descritos a continuación. Sin embargo, la mayoría de los pasos ya han sido realizados. Lo que cambia es recompilar el sistema de archivos para incluir udev: que es el gestor de dispositivos de manera dinámica. También es nuevo el crear el archivo S51hostapd.

Configurar la red Wifi

Es necesario que se seleccione el udev desde buildroot. udev es un administrador de dispositivos que asigna de manera dinámica los drivers de dispositivos y evita que uno tenga que hacer la configuración manualmente.
ir a */buildroot-2013.08.1.*
make clean
make menuconfig
System Configuration --> /dev management --> Dynamic using udev
make

NOTA IMPORTANTE: como udev administra dinámicamente los drivers, sólo crea los archivos cuando los necesita. Por lo tanto, el sistema de archivos que incluya udev tiene la carpeta /dev prácticamente vacía. Hay que extrar en la microSD el sistema de archivos primero sin udev (de manera que se carguen los drivers que tienen soporte) y después sí extraer el sistema de archivos con udev.
Agregar carpetas y archivos faltantes. Copiar la carpeta *filesystem_mini_network_client* dentro de la microSD.
revisar y configurar los siguientes archivos:
/etc/default
isc-dhcp-server
INTERFACES="wlan0"
/etc/hostapd
hostapd.conf

interface=wlan0
driver=nl80211
ssid=NOMBRE_RED
channel=1
hw_mode=g
auth_algs=1
#wpa=3
#wpa_passphrase=siebot_2013
#wpa_key_mgmt=WPA-PSK
#wpa_pairwise=TKIP CCMP
rsn_pairwise=CCMP
/etc/network
interfaces

Configure Loopback
auto lo
iface lo inet loopback

auto wlan0
iface wlan0 inet static
address 192.168.50.1
netmask 255.255.255.0
gateway 192.168.50.1
wireless_essid NOMBRE_RED
wpa-ssid NOMBRE_RED
wpa-psk 1011215410
wpa-conf /etc/wpa_supplicant.conf
/etc/dhcp
dhcpd.conf

subnet 192.168.50.0 netmask 255.255.255.0 {
range 192.168.50.5 192.168.50.20;
option domain-name-servers 192.168.50.1, 192.168.50.1;

option routers 192.168.50.1;
}
/etc/
wpa_supplicant.conf

ctrl_interface=/var/run/wpa_supplicant
#ap_scan=1

network={
ssid="NOMBRE_RED"
scan_ssid=1
key_mgmt=WPA-PSK
psk="1011215410"
proto=RSN WPA
pairwise=CCMP TKIP
group=CCMP TKIP
}
/_
make_snd
Eliminar todos los *sudo's* que haya. La stamp no identifica ese comando a menos que en el sistema de archivos le hayan dado soporte.
Los valores en azul dependerán del nombre y la dirección que cada grupo quiera para su tarjeta.
Crear script de inicio
ir a */etc/init.d*
crear un archivo llamado S51hostapd con la siguiente línea
/usr/sbin/hostapd /etc/hostapd/hostapd.conf &

Este script se ejecuta al iniciar el sistema y configura la red como access point.

Muchachos, una aclaración. La carpeta *filesystem_mini_network_client* en donde están los archivos que hay que modificar NO la crea el sistema. Esa carpeta la envió Camargo. En caso de que les falte algún archivo o carpeta lo pueden crear como texto plano y poner lo que dice ahí.

Otra aclaración.

es necesario escribir la siguiente línea en la consola del computador.

apt-get install dhcp3- server dhcp3-client

Esto va a permitir que la red que creamos asigne direcciones de manera dinámica. Sin embargo, esto no está funcionando bien entonces hay que hacerlo, por ahora, de manera manual. Para ello hay que hacer lo siguiente:

Una vez se conecten a la tarjeta desde, por ejemplo, el computador deben editar las conexiones inalámbricas (en la barra de arriba donde salen las conexiones de red --> Edit connections --> Wireless --> Seleccionan su red (NOMBRE_RED) --> Edit). Buscan la pestaña IPv4 Settings y en Method seleccionan Manual. Agregan una dirección y ponen lo siguiente.

Addresses
Address Netmask Gateway
192.168.50.2 192.0.0.0 192.168.50.1

DNS servers: 192.168.50.1

El color en rojo es la dirección de su computador en el servidor de la tarjeta.

Eso siempre se demora un poquito en conectar.

Para ver (desde la tarjeta) que la red sí está habilitada deben escribir ifconfig y debería salir wlan0.

con el comando ssh root@192.168.50.1 desde la consola del computador se pueden conectar a la tarjeta (controlar la tarjeta desde el pc). Es posible que en este paso les salga un mensaje de "error" en donde dice que no es seguro conectarse. En este caso hay que ir a buildroot y poner una contraseña al sistema, esa es la contraseña para acceder a la tarjeta. NO es necesario hacer make clean en buildroot para poner la contraseña.

El make_snd se dejó en la carpeta raíz de la SD.

Luego Camargo envió el siguiente correo:

Hola muchachos

Para que puedan correr bien el ejemplo del blink en la tarjeta no es necesario usar boa. boa es un servidor web que utilizamos para colocar páginas web, pero el ejemplo del blibnk que les moestré crea su propio servidor en el puerto 8888

Lo que deben hacer es correr el programa con node:

node blink.js

al ejecutar ese comando se crea el servidor en el puerto 8888

para verificar que esté funcionando bien deben ingresar a la siguiente dirección

http://192.168.50.1:8888
(o la dirección que le pusieron a su tarjeta)

Ya pueden probar la configuración del http, el problema que tenían era elmismo de la falta de asignación de IP, como ya tienen un IP asignado pueden ejecutar:

dhcpd -cf /etc/dhcp/dhcpd.conf wlan0 &

y ahi seguramente ya les da una dirección IP sin tener que ponerla a mano

Y Ricardo, nos envió el siguiente correo:

Buenas noches, hoy estuvimos mirando lo del javascript y relaizamos los siguientes cambios para que el servidor realizara todas las configuraciones de red de forma automática en cualquier dispositivo que se desee conectar para no tener que hacerlo cada vez de forma manual. Lo que hicimos fue lo siguiente:
En la carpeta /etc/init.d/
Modificar el archivo S51HOSTAPD
de la siguiente forma:

/usr/sbin/hostapd /etc/hostapd/hostapd.conf & %%% ESTA ES LA QUE YA ESTÁ, NO REPETIRLA!!

touch /var/lib/dhcp/dhcpd.leases %%% PARA QUE DEJARA CORRER EL DHCPD.CONF FUE NECESARIO CREAR ESTE ARCHIVO

dhcpd -cf /etc/dhcpd.conf wlan0 & %%%VUELVE A HABILITAR LA RED YA QUE CON EL COMANDO

ANTERIOR SE DESMONTA

mkdir /var/log/boa %%% CREA EL BOA PARA PODER ACCEDER A LA TARJETA DESDE EL NAVEGADOR

Con esto ya se pueden conectar directamente desde cualquier computhttp://www.google.com/search?q=192%3A168%3A50%3A1&ie=utf-8&oe=utf-8&client=ubuntu&channel=fsador o celular.
Cualquier cosa me cuentan.

Para crear el AP se ejecuta

$ hostapd /etc/hostapd/hostapd.conf

El Hostapd funciona y crea el AP, pero dhcpd no funciona aún. Conectandose con IP fija a la tarjeta y cargando el ejemplo del blink, se puede correr el ejemplo del blink

$ cd /var/www/
$ node blink.js

El problema que muestra el dhcpd al correr $ dhcpd -cf /etc/dhcp/dhcpd.conf wlan0 &, es: *"Can't open lease database /var/lib/dhcp/dhcpd.leases: No such file or directory".*

Según la documentación:
Dhcpd can be made to use an alternate configuration file with the -cf flag, or an alternate lease file with the -lf flag. Because of the importance of using the same lease database at all times when running dhcpd in production, these options should be used only for testing lease files or database files in a non-production environment.

Se intentó agregar la linea $ dhcpd -lf /var/lib/dhcp/dhcpd.leases &. al archivo S51hostapd.
No funcionó.

Al buscar el archivo desde la consola de la mini-stamp, NO aparece el archivo dhcpd.leases.... no sabemos como hacerlo aparecer. Al parecer el sistema lo borra al reiniciarse.

Funcionó así:
Se eliminan las líneas del S51hostapd, dejando solo las iniciales:

#touch /var/lib/dhcp/dhcpd.leases
#dhcpd -cf /etc/dhcp/dhcpd.conf wlan0 &
#dhcpd -lf /var/lib/dhcp/dhcpd.leases &
#mkdir /var/log/boa

Se movio el .leases a var/lib
Se creó un script (.sh) que hacía lo siguiente:

```
#!/bin/bash
hostapd /etc/hostapd/hostapd.conf &
sleep 5
dhcpd -lf /var/lib/dhcpd.leases &
sleep 5
cp /var/lib/dhcpd.leases /var/lib/dhcp/
dhcpd -cf /etc/dhcp/dhcpd.conf wlan0 &
```

Arranca el hostapd; espera 5s para que arranque; luego cambia la ruta del archivo .leases e inicializa el archivo; espera 5 segundos a que haga esto; copia el archivo .leases a la carpeta /var/lib/dhcp, para que al ejecutar la siguiente instrucción, se encuentre el archivo. Esto se hace

para que al iniciar el sistema, cuando borra el .leases, el script lo vuelva a crear e inicializar.

El script se guardó en el root de la SD, y se ejecuta así:

```
$ cd /;
$ ./network.sh
```

PROYECTO
OSCILOSCOPIO DIGITAL

DESCRIPCIÓN DEL PROYECTO

El objetivo es crear un dispositivo con el cual el estudio de los circuitos y las señales eléctricas sea más intuitivo y natural. Este dispositivo funcionará como voltímetro digital u osciloscopio. Será usado en colegios del distrito para la enseñanza básica en física. Este deberá ser económico y presentar un diseño simple y robusto, para estudiantes de colegio de múltiples edades.

Se quiere usar la plataforma embebida MINI-STAMP como sistema de adquisición de señales eléctricas, y envío de las mismas, por medio de una red LAN, a un dispositivo móvil para su visualización.

CARACTERISTICAS:

- Sensado de señales eléctricas de tensión entre -5 y 5V
- Dos canales de medición
- Uso de sondas de tensión sin atenuación

2. PROYECTOS SIMILARES

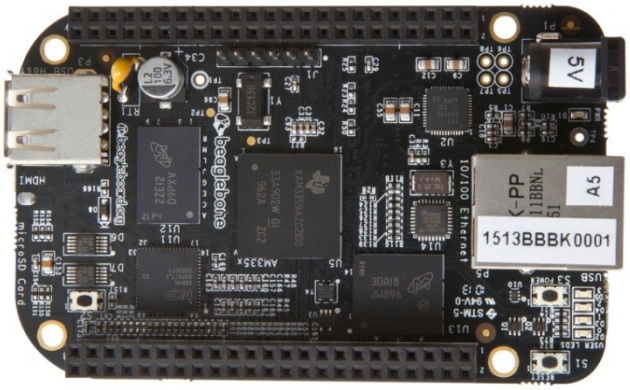
2.1 LabQuest 2

Un ejemplo de plataforma comercial es la llamada LabQuest 2, que es un pequeño sistema capáz de integrar la medición de múltiples variables, visualizarlas en un entorno agradable, almacenarlas y procesarlas para realizar análisis. La plataforma soporta múltiples sensores estándar, como: temperatura, sonido, luz, posición (por ultrasoido), etc.
http://www.vernier.com/products/interfaces/labq2/

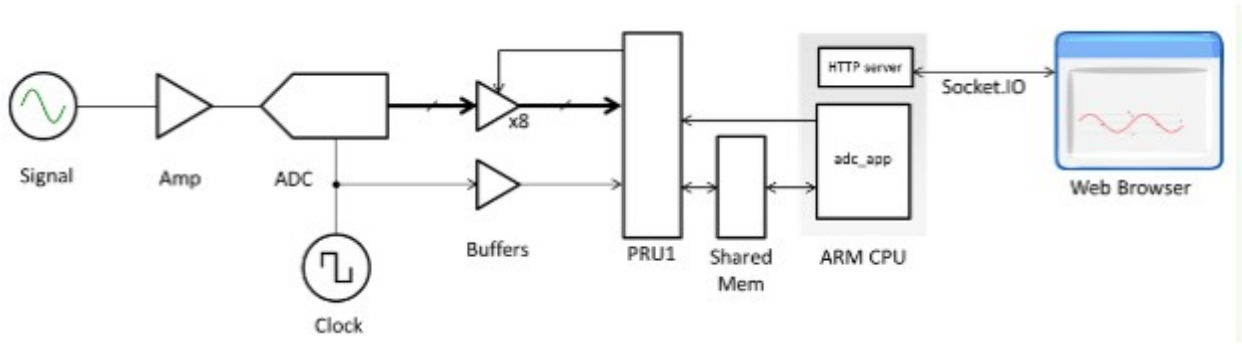
Esta plataforma permite almacenamiento y procesamiento de datos, así como conexión a internet y envío de información. El LabQuest 2 cuesta USD 329.

2.2 BBB - High speed data acquisition and web-based UI

http://www.element14.com/community/community/knode/single-board_computers/next-gen_beaglebone/blog/2013/08/04/bbb--high-speed-data-acquisition-and-web-based-ui



Usando la plataforma Beagle-Bone Black, realizan un sistema de transmisión de datos vía WiFi de alta velocidad que permite sensar una señal con un ADC de alta frecuencia de conversión, y enviar los datos a un dispositivo móvil en el cual son visualizados.



La señal es muestreada por el ADC (que usa un clk externo). A la salida paralela del ADC se usan buffers para mejorar el acoplamiento entre el ADC y la unidad PRU (Programmable Real-Time Unit) de la Beaglebone. Se crea una aplicación en Linux llamada adc_app, que toma la información de la unidad PRU y la almacena en una memoria compartida. Posteriormente esa información es organizada en un archivo csv y enviada vía web usando la tarjeta Beaglebone también como un servidor.

En la implementación del servidor se usa Node.JS, que es un entorno de programación del lado del servidor, y que usa Javascript como lenguaje de programación. Permite entradas y salidas de datos. Usa SocketIO para crear las entradas y salidas de datos. Socket.IO es una librería de Javascript que permite comunicaciones en tiempo real de aplicaciones web.

3. CARACTERÍSTICAS DEL PROYECTO

3.1 Características del ADC del ATSAM3N:

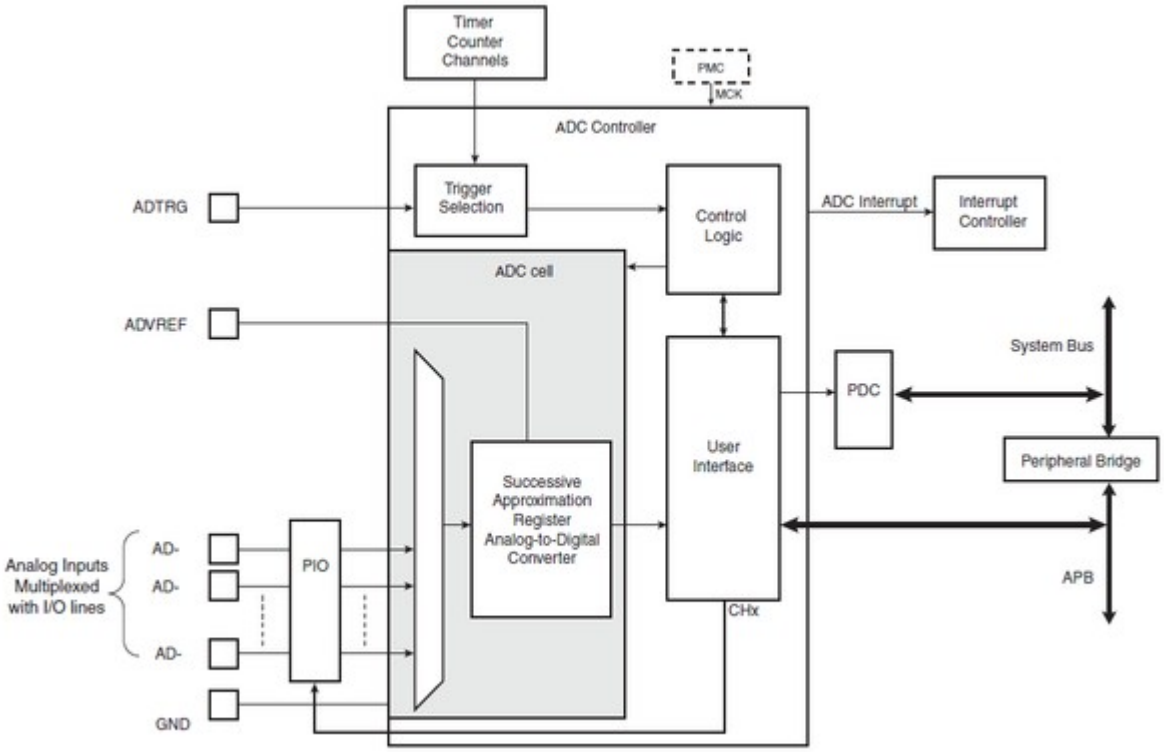
El procesador ATSAM3N0A de Atmel incluye:

- 16 canales multiplexados para ser usados con el ADC.
- El ADC incorporado es de *modo común*, es decir, su referencia de señal es la misma referencia de tierra del circuito (la misma tierra digital).
- Posee un pin de referencia de voltaje ADVREF que puede encontrarse en el rango de 2,6V a

3,6V.

- Resolución de 8 bits o 10 bits configurable por software.
- Registros de almacenamiento de datos individuales para cada canal
- Frecuencia de conversión de hasta 500KHz

El esquema del ADC es presentado a continuación:



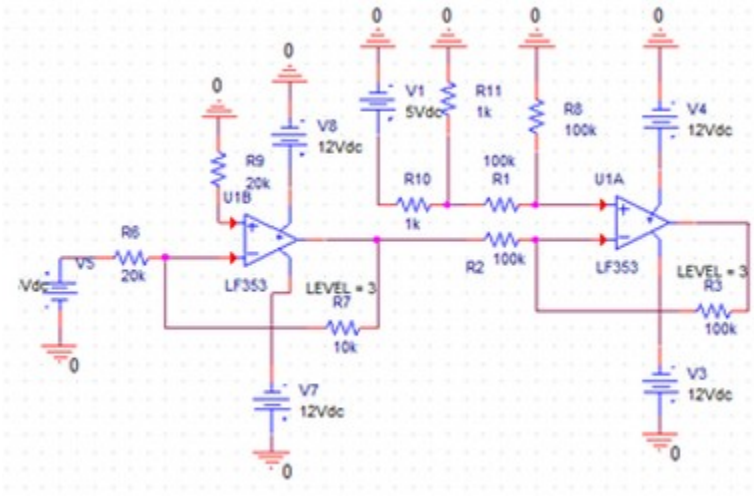
La MINI-STAMP posee un regulador de 3.3V, que sería la tensión de referencia para el ADC en el pin ADVREF.
Claramente este no es un ADC diferencial, por lo que no se puede realizar el sensado de señales negativas.
La diferencia de un ADC diferencial y un ADC común está explicada en:

http://electronics.stackexchange.com/questions/3727/what-is-a-differential-adc

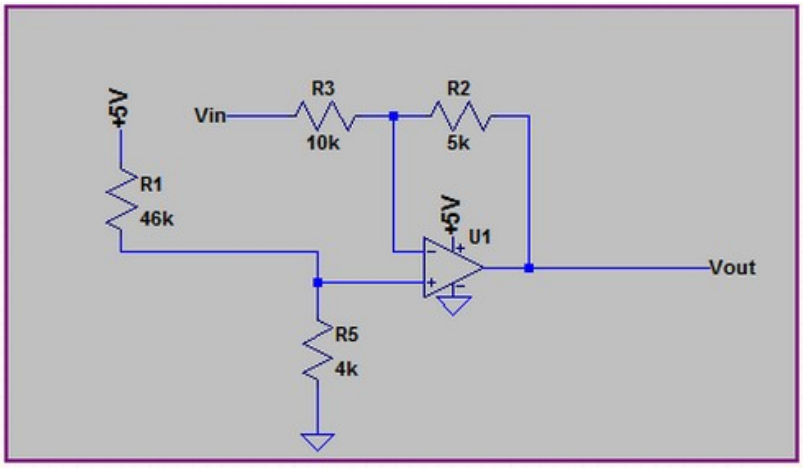
Por tanto se requiere de un acoplamiento de la señal.

3.2 Acoplamiento de la señal*:*

Puesto que la señal a visualizar puede ser bipolar, se requiere realizar un acondicionamiento de esta para que pueda ser leída correctamente por el ADC, tal que no quede por fuera del rango de muestreo del ADC. Este acondicionamiento se puede realizar usando amplificadores operacionales.
OPCIÓN 1: Uso de 2 amplificadores, con polarización bipolar. El primero realiza un escalamiento de la señal, reduciendo su *span* y el segundo inserta un offset para no tener señales negativas. A continuación se muestra:



OPCIÓN 2:
Uso de un solo amplificador operacional, usando una sola fuente de alimentación, en el cual se implementa al tiempo el escalamiento de la señal y el offset que se le debe dar, para evitar señales negativas entrantes al ADC.



Claramente esta opción es mucho más sencilla y produce el mismo efecto requerido. *TENER EN CUENTA:* La impedancia de entrada de la OPCIÓN 2 es R3 (¿¿¿???), por lo que la relación de impedancias de la sonda y la resistencia de entrada es importante para el escalamiento de la señal en el osciloscópio.

Sondas Atenuadas: Poseen una impedancia asociada de aproximadamente 9Mohms. Los osciloscópios comunes poseen una impedancia de entrada de 1Mohm, por lo que la atenuación producida por la sonda (comúnmente x10) es el resultado del divisor de tensión entre las impedancias de la sonda y el osciloscópio. De esta manera, R3 debería ser de 1Mohm para poder usar sondas atenuadas, y así ampliar el rango de tensión posible por el ADC.

4. DESARROLLO

4.1 Pruebas con el ADC

Para poder usar el SAM3N, este requiere de un sistema operativo que lo controle. Se llama ChibiOS, que es un OS muy liviano y orientado a la programación de aplicaciones en tiempo real. Permite crear hilos de procesos paralelamente, y manejar todos los periféricos del SAM3N.

La aplicación que se desee implementar con el SAM3N se escribe en código C, y usando la cadena de herramientas de compilación para el SAM, se genera un archivo binario ch.bin, que es el que contiene tanto el OS que va a controlar el procesador, como el código de la aplicación que se desea correr.

Para pasar este archivo a la memoria del SAM, primero se copia en alguna parte de la SD, y luego se usa la aplicación “openocd” para enviar vía puerto serial, el ch.bin desde la memoria SD, a través del iMX, hasta la memoria del SAM.

La aplicación Openocd fue incluida dentro del sistema de archivos cuando fue creado.

Para el caso del uso del ADC, el ejemplo se encuentra en /ChibiOS_mini/demos/ARMCM3-SAM3N-EK/main.c; en esa carpeta también se encuentra el Makefile, así como todos los demás archivos y librerías provistas por Atmel para usar las múltiples funciones preprogramadas. Para compilar el código del ADC junto con el OS: $ make. Esto crea el ch.bin, que se pasa a la SD, junto con múltiples archivos de configuración del SAM, también presentes en la carpeta del main.c

Desde la terminal de la Mini-stamp, en la ubicación del ch.bin:

$ openocd -f stamp_mini_sam3n.cfg

Para crear el nodo de comunicación entre el SAM y el iMX, que va a ser un puerto serial:

$ mknod /dev/ttySP1 c 242

El “c” significa: comunicación tipo char: envío de datos por carácter, no por bloques.
El puerto serial está por defecto configurado para una velocidad de transmisión de 38400 baudios, pero el ejemplo del ADC transmite a 115200 baudios. Para cambiar esto

$ stty -F /dev/ttySP1 115200

Ahora sí, se puede ver lo que está recibiendo el puerto:

$ cat /dev/ttySP1

El proceso se corta con ctrl + C.

4.2 CAPTURA Y ALMACENAMIENTO DE DATOS

Se desea desarrollar un programa en código C que lea la información del puerto serial, y la almacene en un archivo en formato .json, que luego será importado por Node.js para ser graficado en el navegador. La info que está siendo enviada por el puerto serial son los datos del ADC del SAM3N.

El núcleo de este código está en los parámetros de configuración del puerto. Se debe incluir la librería <termios.h>, que contiene la estructura "termios" con todos los parámetros de configuración del puerto. Para nuestra aplicación los parámetros que se deben cambiar son: la velocidad de transmisión (115200 baudios), si hay o no bit de paridad (sí hay), y la cantidad de bits a transmitir (8 en nuestro caso).

Luego, se lee iterativamente el archivo asociado al puerto serial, en el cual se encuentran los datos entrantes en un búffer. Estos datos se van organizando según la sintaxis del formato de archivos .json (www.json.org) en forma de objetos, en un archivo de registro que posteriormente será leido por Node.js.

4.3 JAVASCRIPT

Una vez en la tarjeta posee un servidor web es momento de enfocarse en la creación de una

página web donde se muestren los datos obtenidos con la tarjeta.

Para poder realizar la visualización de la información en algún dispositivo se utilizó Javascript. Este es un lenguaje de programación orientado a objetos que es comúnmente utilizado dentro de la programación de páginas web cuyo código es generalmente ejecutado en el navegador. Para poder utilizar Javascript por fuera del navegador se utiliza Node.js, un entorno de programación orientado a eventos basado en Javascript, que en lugar de ejecutarse en el navegador se ejecuta del lado del servidor. Además de ser un entorno, Node.js también es utilizado como librería de Javascript.

Al utilizar Node.js lo que se hace es implementar un servidor HTTP, de tal manera que nuestra aplicación pueda ser observada por el usuario y que esta sea capaz de responder a las diferentes peticiones que pueden surgir de la interacción del usuario con la página web.

Para lograr esto, la aplicación se va a dividir en 3 programas Javascript diferentes, el primero (generalmente llamado*server.js*) nos va a permitir implementar e iniciar el servidor, especificando una dirección IP y un puerto de acceso, y además va a recibir las diferentes peticiones del usuario. El segundo archivo (generalmente llamado *router.js*) se va a hacer cargo de direccionar las peticiones recibidas, asignando la petición al manejador correspondiente o avisando al usuario que la petición realizada no posee ninguna manejador que pueda tratarla. Finalmente el tercer archivo (generalmente llamado *requestHandler.js*) se encarga de tratar las peticiones que le son enviadas, es en este archivo donde, según la petición que se realice, se especifica el código HTML que es el que realiza la interfaz gráfica de la página web a partir de la cual el usuario esta interactuando con la aplicación. En este archivo no solo se tratan las diferentes peticiones sino que se definen los diferentes manejadores que son usados en la aplicación.

Cabe resaltar que esta no es la única forma en que una aplicación web se puede desarrollar, el formato que se adoptó está basado en lo tratado por el libro *The Node Beginner Book,* escrito por *Manuel Kiessling.*
(ver*http://www.nodebeginner.org*)

La primera tarea a realizar era la creación del archivo *server.js,* en esta se definía la dirección IP y el puerto en el q si iba a trabajar. La dirección se eligió de acuerdo con la que ya había sido seleccionada durante la configuración de los diferentes archivos de la tarjeta para habilitar la red. El puerto elegido fue el 8888, ya que en sistemas operativos basados en Linux no es posible hacer enlaces a puertos menores de 1024 a menos que se habiliten ciertas opciones para las cuales el kernel de la tarjeta no tiene soporte.

```
var http = require("http");

var url = require("url");



function iniciar(route, handle) {

  function onRequest(request, response) {

    var pathname = url.parse(request.url).pathname;

    console.log("Request for " + pathname + " received.");
```

```
    request.setEncoding("utf8");


    request.addListener("end", function() {
      route(handle, pathname, response);
    });


  }
  http.createServer(onRequest).listen(8888);
  console.log("Servidor Iniciado.");
}

exports.iniciar = iniciar;
```

A continuación se creó el archivo *router.js,* cuyo rol ya fue explicado anteriormente y cuyo código se muestra a continuación.

```
function route(handle, pathname, response) {
  console.log("A punto de rutear una peticion para " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response);
  } else {
    console.log("No se ha encontrado manipulador para " + pathname);
    response.writeHead(404, {"Content-Type": "text/html"});
    response.write("404 No encontrado");
    response.end();
  }
}

exports.route = route;
```

Finalmente se encuentra requestHandler.js, que es el archivo más extenso y el que más complicaciones puede traer, ya que en este es que se describen las funciones que se deben realizar según las solicitudes realizadas y además se describe el código HTML que define lo que se va a mostrar en la página web con la cual el usuario va a interactuar. La aplicación a realizar se

basa básicamente en hacer la toma de datos a través de la tarjeta y poder graficar los resultados obtenidos, razón por la que se debía hacer énfasis en aprender a graficar y a leer datos de archivos preestablecidos.

Para la parte de realizar gráficas existen dos opciones básicas que se pueden utilizar de la librería *jquery:* las librerias *Flot*y *Smoothie.* Estas dos librerías permiten crear gráficas que se van a ubicar en marcos creados anteriormente, para la aplicación se decidió utilizar *Flot.js* porque, si bien las dos son fáciles de utilizar, ésta librería posee mayor documentación sobre cómo realizar gráficas cuando los datos se encuentran en archivos externos al código.

Para la parte de adquisición de datos se decidió que los resultados obtenidos en la tarjeta se guardarían en un archivo*.json*. Usando la librería *jquery* existe la función *Ajax,* la cual puede tomar la dirección de un archivo de diversos tipos y retornar su contenido para que este sea tratado en la aplicación. Este fue el procedimiento a realizar, sin embargo en este caso el archivo a leer se encontraba en la dirección *'http://192.168.50.1'*, mientras que la aplicación se encuentra en la dirección *'http://192.168.50.1:8888".*

Cuando las direcciones de un archivo que desea ser leído utilizando *Ajax* no se encuentra en la dirección base donde la función es utilizada se produce un error de dominios cruzados llamado " Access-Control-Allow-Origin", que básicamente no permite por motivos de seguridad abrir archivos de otras direcciones web. Para solucionar este inconveniente se debe agregar un archivo llamado "*.htaccess'* a la dirección donde se encuentra el archivo a ser leído, para convertir el error en flag, y además se debe ejecutar la aplicación en una ventana de Google-Chrome donde se hallan deshabilitado ciertos elementos. El contenido del archivo *.htaccess* es:

```
Header set Access-Control-Allow-Origin "*"
```

Para lograr esto se escribe en terminal:

```
$ google-chrome --disable-web-security
```

Volviendo al archivo *requestHandler.js*, luego de haber solucionado los problemas expuestos, en este archivó se diseñó una página principal que permite acceder a la gráfica resultante mediante un botón y esta página donde se muestra la señal lleva otra página que permite volver al inicio. El código de este archivo se muestra a continuación:

```
var fs     = require('fs');
var server = require("./server");
var router = require("./router");
var ADC_DEV_NAME = '/proyecto/salida';

function iniciar(response) {
  console.log("Manipulador de Peticion 'iniciar' fue llamado.");
  fs.writeFile(ADC_DEV_NAME, "Y" );

  var body = '<html>'+
```

```
    '<head>'+
    '<title>Digital Oscilloscope - Embedded Systems</title>'+
    '<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />'+
    '<link href="http://192.168.50.1/lib/examples.css" rel="stylesheet"
type="text/css">'+
    '<script language="javascript" type="text/javascript"
src="http://192.168.50.1/lib/jquery.js"></script>'+
    '<script language="javascript" type="text/javascript"
src="http://192.168.50.1/lib/jquery.flot.js"></script>'+
    '</head>'+
    '<body>'+
    '<div id="header">'+
    '<h3>Osciloscopio Digital - Sistemas Embebidos</h3>'+
    '</div>'+

    '<div id="content">'+

'<b>Presentaci&oacuten</b>'+
'<br/>'+
'<br/>'+

'<p>Mediante esta aplicaci&oacuten se pretende facilitar la realizaci&oacuten
de pr&aacutecticas de laboratorio de F&iacutesica en colegios.'+
' Esta p&aacutegina funciona como un osciloscopio digital, la tarjeta a la cual
est&aacute asociada la p&aacutegina se encarga de realizar'+
' la adquisici&oacuten de datos y el env&iacuteo de los mismos para que sean
graficados en este lugar.</p>'+
'<br/>'+

'<p>Por favor conecte el dispositivo de medici&oacuten a los terminales del
circuito que desea probar. Una vez haya verificado que las '+
'conexiones realizadas son correctas y que los valores esperados a la salida no
sobrepasan los rangos de operaci&oacuten del dispositivo '+ 'oprima el
bot&oacuten <b><i>Graficar Salida</i></b> para que la adquisici&oacuten de
datos comience. Este bot&oacuten lo llevara a una nueva '+ 'p&aacutegina donde
usted podr&aacute observar la señal de salida medida de su circuito.</p>'+
'<br/>'+

    '<p>'+
    '<form action="/subir" method="post">'+
    '<input type="submit" value="Graficar Salida" FONT-SIZE= 12px />'+
    '</form>'+
    '</p>'+
    '</div>'+
    '</body>'+
```

```
    '</html>';

response.writeHead(200, {"Content-Type": "text/html"});
response.write(body);
response.end();
}


function subir(response,output) {
  console.log("Manipulador de Peticion 'subir' fue llamado.");
  console.log("Taking Samples from ADC: ");
  fs.writeFile(ADC_DEV_NAME, "Y" );

  var body = '<html>'+
'<head>'+
'        <title>Results: Graphs</title>'+
'        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">'+
'        <link href="http://192.168.50.1/lib/examples.css" rel="stylesheet"
type="text/css">'+
'        <script language="javascript" type="text/javascript"
src="http://192.168.50.1/lib/jquery.js"></script>'+
'        <script language="javascript" type="text/javascript"
src="http://192.168.50.1/lib/jquery.flot.js"></script>'+
'        <script type="text/javascript">'+

'        $(function() {'+

'                var options = {'+
'                        lines: {'+
'                                show: true'+
'                        },'+
'                        points: {'+
'                                show: true'+
'                        },'+
'                        xaxis: {'+
'                                tickDecimals: 0,'+
'                                tickSize: 1'+
'                        }'+
'                };'+

'                var data = [];'+
'                $.plot("#placeholder", data, options);'+

               // Fetch one series, adding to what we already have
'                var alreadyFetched = {};'+
```

```
'                $("button.fetchSeries").click(function () {'+

'                        var button = $(this);'+

                       // Find the URL in the link right next to us, then fetch
the data
'                        var dataurl = button.siblings("a").attr("href");'+

'                        function onDataReceived(series) {'+

                                // Extract the first coordinate pair; jQuery
has parsed it, so
                                // the data is now just an ordinary JavaScript
object
'                                var firstcoordinate = "(" + series.data[0][0]
+ ", " + series.data[0][1] + ")";'+
'                                button.siblings("span").text("Fetched " + se-
ries.label + ", first point: " + firstcoordinate);'+

                                // Push the new data onto our existing data ar-
ray
'                                if (!alreadyFetched[series.label]) {'+
'                                        alreadyFetched[series.label] = true;'+
'                                        data.push(series);'+
'                                }'+

'                                $.plot("#placeholder", data, options);'+
'                        }'+

'                        $.ajax({'+
'                                url: dataurl,'+
'                                type: "GET",'+
'                                dataType: "json",'+
'                                success: onDataReceived'+
'                        });'+
'                });'+

        // Load the first series by default, so we don't have an empty plot
'                $("button.fetchSeries:first").click();'+
'        });'+
'        </script>'+
'</head>'+

'<body>'+
```

```
'        <div id="header">'+
'                <h2>Resultados - Gr&aacuteficas</h2>'+
'        </div>'+

'        <div id="content">'+

'                <div class="demo-container">'+
'                        <div id="placeholder" class="demo-
placeholder"></div>'+
'                </div>'+

'                <p>'+

'                        <button class="fetchSeries">Graficar</button>'+
'                        [ <a href="http://192.168.50.1/data.json"
target="_blank">        see data</a> ]'+
'                        <span></span>'+
'                </p>'+

'                <p>'+
'                        <button class="back">Actualizar Datos</button>'+
'                        [ <a href="/salir">reload</a> ]'+
'                </p>'+
'        </div>'+

'</body>'+
'</html>';
response.writeHead(200, {"Content-Type": "text/html"});
response.write(body);
response.end();
}


function salir(response) {
  console.log("Manipulador de Peticion 'iniciar' fue llamado.");
  fs.writeFile(ADC_DEV_NAME, "Q" );
  subir(response);
}

var handle = {}
handle["/"] = iniciar;
handle["/iniciar"] = iniciar;
handle["/subir"] = subir;
handle["/salir"]=salir;
```

```
server.iniciar(router.route, handle);
```