Revisión	Fecha	Comentario	Autor
0	03/02/2011	Display: Ampire AM320240L8-TNQWTB1H Módulos: Digi CC9P/W9P S.O.: NET+OS 7.4.2.	Ulises Bigliati

Índice

El display color

Vamos a utilizar un módulo LCD[1] color TFT con controlador incorporado[2], con una resolución de 320x240x16/18 bits. (65K o 262K colores seleccionable mediante un pin de configuración)
La interfaz de nuestro display consiste en un bus de datos que puede usarse en 18, 16, 9 u 8 bits, mas las señales de control I80 o M68.
El back-light es del tipo LED, y su brillo puede ser controlado desde la línea de entrada *LED_A* con una señal PWM de 100Hz a 1KHz, siendo que con un ciclo de trabajo de 0% (siempre 1) el brillo es el máximo. La alimentación del back-light está resuelta internamente de modo que solo es necesario proveerle el display la tensión de alimentación principal de 3.3V, la especificación de consumo del fabricante es de 320mA y es además es 5V-tolerant en las entradas */CS*, */WR*,*/RD*,*RS*,*DB0~DB17*.
El módulo también incluye un panel de touch screen resistivo y su respectivo controlador[3] que presenta su interfaz de comunicación SPI a través del pin-out del módulo.
El pin-out[4] de la interfaz es accesible mediante un conector FFC/FPC de 40 contactos de paso 0.5mm.

La plataforma de desarrollo

Asumimos que estamos trabajando sobre un kit de desarrollo de Digi para sus módulos ConnectCore 9P/W9P. En cuanto al módulo, este ofrece la posibilidad de acceder a un bus de periféricos externos de 16 bits con interfaz tipo I80, es decir, que disponemos de las líneas de control que requiere nuestro display.
Este bus puede ser accedido cómodamente durante las pruebas gracias a la placa de prototipos incluida en los kits de Digi.

---

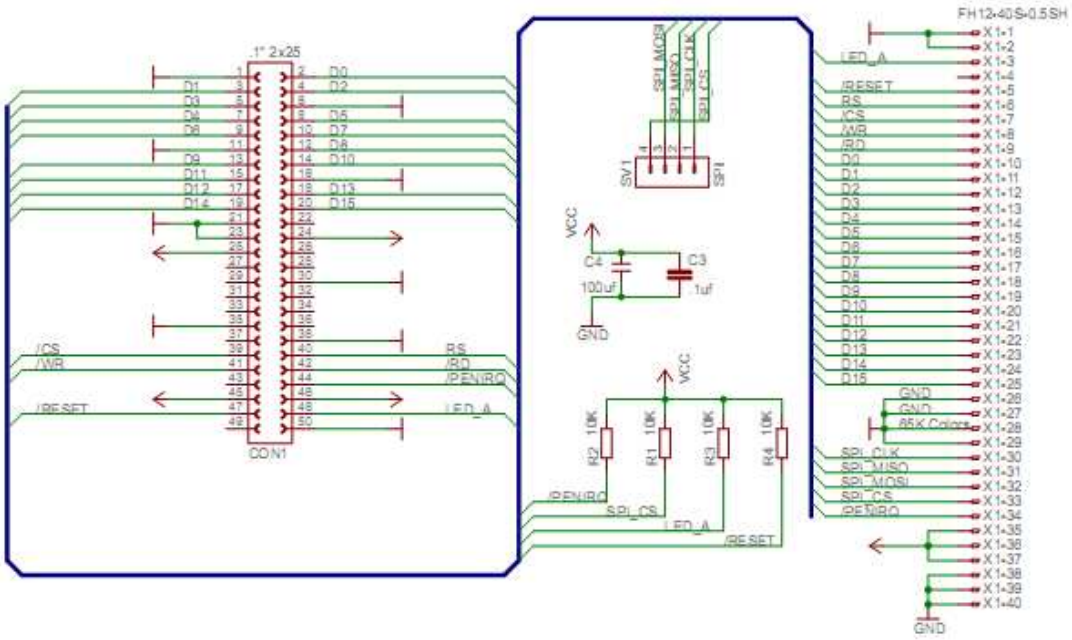
[1] El display utilizado es de la marca Ampire, modelo AM320240L8-TNQWTB1H.
[2] El controlador que está integrado en el display es el FSA506 (ver hojas de datos).
[3] El controlador de touch screen es el TSC2046 (ver hojas de datos).
[4] El pin-out del módulo está disponible en la hoja de datos que acompaña a este trabajo (AM320240L8TNQW-TB4H.pdf, página 13).

En el conector X33 de 50 pines[5] (application header) se exponen las señales necesarias según el pin-out que podemos apreciar en la tabla:
- El bus de datos de 16bits (BD0-BD15)
- Las señales de control para el bus: CS (*EXT_CS0*), WE(*EXT_WE*), OE(*EXT_OE*).
- El bus de direcciones (BA0-BA9), (el cual no usaremos).
- Algunos pines de uso general, a saber:
- GPIO103, que será usado como la señal RS[6].
- GPIO101, utilizado como línea de interrupción para el touch screen.
- GPIO86, utilizado como señal del /Reset.
- GPIO87, destinado a activar el back-light y potencialmente utilizable como señal PWM.
- La alimentación de 3,3Volts.

Lamentablemente no contamos con las señales del puerto SPI (CE,DI,DO,CLK) directamente en X33 para la conexión del controlador de touch screen.
Para obtener estas señales tendremos que recurrir al conector X8 de la placa de prototipos[7].

Interfaz de conexión entre el display y la placa de prototipos

Habiendo ya presentado las interfaces que nos presentan el microprocesador y el display, nos resta diseñar la interconexión entre ambos. Para esto diseñamos el esquemático y a partir de este un pequeño PCB.



Aquí, el conector CON1 se corresponde con X33 y se conecta directamente en este, el FH12... es el conector para el cable plano flexible del display, y SV1 es el conector destinado a llegar hasta X8, cable mediante.

[5] Pueden verse los detalles en la documentación del fabricante (9P_HR.PDF, página 63).
[6] Si bien en ocasiones es posible aprovechar alguna línea del bus de direcciones para generar esta señal, en nuestro caso no fue posible dado que no cumplía con los requerimientos de los tiempos de acceso a nuestro display.
[7] Página del documento 9P_HR.PDF, página 54.

Configuración del proyecto de NET+OS 7.4.2[8]

Para poder utilizar correctamente tanto el display como el touch screen, será necesario realizar algunos preparativos previos sobre un proyecto nuevo de NET+OS. Para la realización de este trabajo utilizamos el entorno de desarrollo NET+OS, en su versión 7.4.2. pero de todas maneras, el código fuente realizado puede ser portado sin inconvenientes a otras versiones del NET+OS, siempre y cuando, la plataforma de hardware ConnectCore 9P 9215 esté disponible en dicha versión.

- Desde el IDE del NET+OS debemos crear un proyecto nuevo (File -> New -> NET+OS Project) y configurarlo con sus opciones por default.
- Una vez abierto el nuevo proyecto, debemos modificar solo dos archivos del BSP:
    - bsp.c
    - customizeCache.c

bsp.c

En este archivo, debemos ubicar la table `NAStaticMemoryTable` y modificar el registro correspondiente para configurar el modo de funcionamiento del Chip Select CS0. Solo el código resaltado en negrita es el que cambia. Debe quedar como sigue:

```
NAStaticMemoryType const NAStaticMemoryTable[] =
{
      {
            1,    // enabled
            0x81,// configuration register
            0x00, // WaitWen Write Enable Delay Register (clock=13.75ns)
            0x00, // WaitOen Output Enable Delay Register
            0x0d, // WaitRd  Read Delay Registers
            0x00, // WaitPage Page Mode Read Delay Register
            0x0b, // WaitWr Write Delay Registers
            0x00, // WaitTurn Static Memory Turn Round Delay Register
            0x40000000, // BaseAddess for chip selects
            0xF0000001  // Mask for the chip select
      }, /*cs0*/

      {1, 0x81, 0x00, 0x01, 0x08, 0x00, 0x08, 0x02, 0x0, 0x0}, /* cs1 */
      {0, 0x181, 0x00, 0x00, 0x1f, 0x1f, 0x1f, 0x0f, 0x03100000,
       0xffff8001},                                              /* cs2 */
      {0, 0x82, 0x02, 0x02, 0x09, 0x02, 0x09, 0x02, 0x0, 0x0}  /* cs3 */
};
```

Nótese que sobre esta tabla estamos definiendo los valores de operación de los registros del controlador de memoria del microprocesador para actuar sobre el bus utilizando CS0. Básicamente, estamos definiendo el uso de un bus de datos de 16 bits y no utilizar el modo "ExtendedWait" (**`0x81`**); y definimos además los delays para los tiempos de acceso de escritura (**`0x0b`**) y lectura (**`0x0d`**). Estos retardos están expresados en ciclos de clock del controlador de memoria (AHB clock = 149.9136 MHz / 2 = 74.9568 MHz) [9]. Para quienes deseen conocer mayores detalles pueden consultar el documento adjunto a esta nota[10] .

---

[8] El material de referencia para estas operaciones consiste en la nota de aplicación "Edt_Display_App_Note.pdf" que se encuentra en el directorio de instalación del NET+OS 7.4.2. Por ejemplo: *"C:\netos74\application_packages\edt_display".*

[9] Ver hoja de datos del módulo CC9P: "9p_hr.pdf", página 27.

[10] "PrimeCell MultiPort Memory Controller (PL172) Technical Reference Manual" Section 3.1. (PrimeCell Memory Controller PL172 .pdf)

customizeCache.c

Finalmente, buscamos el archivo customizeCache.c y lo modificamos de acuerdo al texto que puede verse en negrita, el comienzo de la tabla `mmuTable` debe verse como sigue, y el resto permanece sin cambios.

```
mmuTableType mmuTable[] =
{
/*   Start       End          Page Size          Cache Mode
User Access     Physical Address */
/*==========  ==========   ================   =============
===========     =============== */
    {0x00000000, 0x00ffffff,  MMU_PAGE_SIZE_1M,  MMU_WRITE_BACK,
MMU_CLIENT_RW,  0x00000000, 0x00ffffff}, /* 16 Megs of RAM */

    {0x40000000, 0x47ffffff,  MMU_PAGE_SIZE_1M,  MMU_NONBUFFERED,
MMU_CLIENT_RW,  0x40000000, 0x47ffffff}, /* CS0 - LCD */

    {0x50000000, 0x57ffffff,  MMU_PAGE_SIZE_1M,  MMU_BUFFERED,
MMU_CLIENT_RW,  0x50000000, 0x57ffffff}, /* 8 Megs of flash */
```

La API de control del display

A partir de este punto tenemos el proyecto “en blanco”, listo para incluir el código fuente que permitirá utilizar nuestro display con un simple programa de demostración que ejemplifica cómo utilizar la API.
Nos resta siplemente copiar los siguiente archivos:

- ampire_cfg.h
- ampire_cfg.c
- ampire_utils.h
- ampire_utils.c
- ampire_graphics.c
- ampire_graphics.h
- touch.h
- touch.c
- fonts.h
- bitmap8.h
- bitmap16.h
- root.c

en el directorio principal de nuestro proyecto, sobreescribiendo el archivo original root.c

La funcionalidad de estos módulos se describe muy brevemente a continuación.

Módulo ampire_cfg.c

Configuración e inicialización del display

Aquí tenemos la función de inicialización:
`void amp_init(void);`
La de reset:
`void amp_reset(void);`
Una función para rotar la imágen de acuerdo a la orientación física que le demos al display:
`void amp_rotate(lcd_rotate_t deg);`

Donde `deg` es uno de los siguientes valores: ROT_0, ROT_90, ROT_270, ROT_180
Y finalmente un par de macros para encender o apagar el display, ya que no hemos implementado el control de brillo por PWM (será la próxima): `AMP_BACKLIGHT_ON` y `AMP_BACKLIGHT_OFF`.

Módulo ampire_utils.c

Utilidades de bajo nivel del display, sus funciones son mayormente utilizadas por el módulo gráfico de mas alto nivel y prácticamente no tienen relevancia a nivel de aplicación. Sus funciones son:

Módulo ampire_graphics.c

Utilidades gráficas de alto nivel. Contiene las siguientes utilidades:

Define algunos los colores básicos: BLACK, RED, GREEN, YELLOW, BLUE, WHITE, ORANGE
Define el tipo point_t utilizado en algunas funciones.

Expone las siguientes utilidades gráficas:

```
void amp_line(point_t *a, point_t *b, unsigned short color);
void amp_circle(point_t *p, int r,  unsigned short color, int solid );
void amp_rectangle(point_t *a, point_t *b, unsigned short color, int
solid);
void amp_fillscreen(unsigned short color);
void amp_draw_box(unsigned int x, unsigned int y, int size, unsigned
short color);
int amp_GetStoredImage(unsigned short *image, int size);
void amp_draw_pixel(point_t *p, unsigned short color);
void amp_draw_image(unsigned int left, unsigned int right, unsigned int
top,unsigned int bottom, unsigned short *colors);
void amp_draw_bitmap(unsigned char *bitmap);
```

Incluye algunas funciones de demostración:

```
void amp_color_test(unsigned short * image);
int amp_rainbow_fps_test(int frames);
```

Se declaran los tipos de fuente disponibles: `FONT_SIZE_5x7`, `FONT_SIZE_8x8`, `FONT_SIZE_8x15` y el margen utilizado para escribirlas: `FONT_MARGIN`

Y finalmente la función de escritura de texto:
Cabe mencionar que para escribir con `backcolor` transpartente, este parámetro debe ser el mismo que el parámetro `fontcolor`.

```
void amp_write_at(unsigned int x, unsigned int y, char * t, unsigned
short fontcolor, unsigned short backcolor, int size);
```

Módulo con bitmaps de muestra (bitmap8.h, bitmap16.h)

Se incluye varios bitmaps en 8bpp y 16bpp que son desplegados por la aplicación de prueba con la función `amp_draw_bitmap();` de esta forma es muy sencillo incluir nuevos bitmaps en la aplicación.
Los bitmaps están definidos como `unsigned char []`.

Módulo de tipografías (fonts.h)

Las tipografías disponibles están incluidas en fonts.h, bajo la definición de `unsigned char []`, con un formato rasterizado.

Módulo touch.c

En touch.h se define el pin destinado a recibir la interrupción del touch screen:

```
#define AMP_DEFAULT_TOUCH_PIN       101
```

Y se declara una función callback que será la que reciba los resultados del proceso de atención a dicha interrupción. Así se define su forma:

```
typedef void(*AMP_touch_callback)(int x, int y, int up);
```

Por lo tanto debemos definir una función de acuerdo a dicho formato en nuestro programa principal, allí recibiremos los parámetros de utilidad para hacer uso del touch screen.

Por último, la siguiente es la función de inicialización del driver del touch screen:

```
int amp_setupTouch(void *callback, int touch);
```

Se le debe pasar el puntero a nuestra función callback, es decir, su nombre, y el pin para la interrupción, es decir, `AMP_DEFAULT_TOUCH_PIN`.

Módulo root.c

Es el archivo principal de la aplicación, y aquí se prueban casi todas las utilidades gráficas dentro de un loop infinito. En cuanto al touch screen, una cruz se dibuja en la coordenada correspondiente al pulsar sobre la pantalla.

Conclusiones

El trabajo realizado está enfocado en la conexión de un display específico con un módulo Digi CC9P mediante su bus de periféricos externo, no obstante, el código fuente que se ha escrito puede ser perfectamente reutilizado para controlar otros tipos de display's gráficos que incluyan controlador siempre y cuando se adapten las funciones de acceso en bajo nivel e inicialización de acuerdo con el controlador existente en el nuevo display a utilizar.

En cuanto al módulo de control del touch screen, este es totalmente independiente del driver del display y puede ser utilizado para controlar cualquier panel de touch conectado a un controlador TSC2046 mediante el puerto SPI, incluso si el controlador de touch fuera diferente, los cambios a realizar sobre este driver serían mínimos.

Por otra parte, no menos importante es el hecho de que de esta nota puede abstraerse el método de conexión y acceso físico y lógico al bus de datos, del módulo CC9P, ya sea para el uso ejemplificado o para la conexión de cualquier otro periférico.