Laboratorio 10 – Keil y emWin

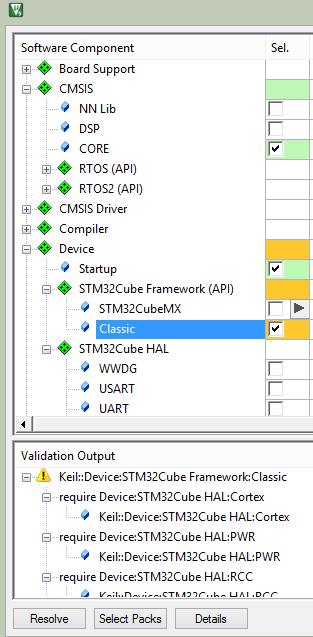
Parte I – GUI con Primitivos

1 – Se debe instalar los siguientes packs: **Keil::STM32F4xx\_DFP**, **Keil::MDK-Middleware**, **ARM::CMSIS**, **Keil::ARM\_Compiler**

2 – Cree un nuevo proyecto en Keil

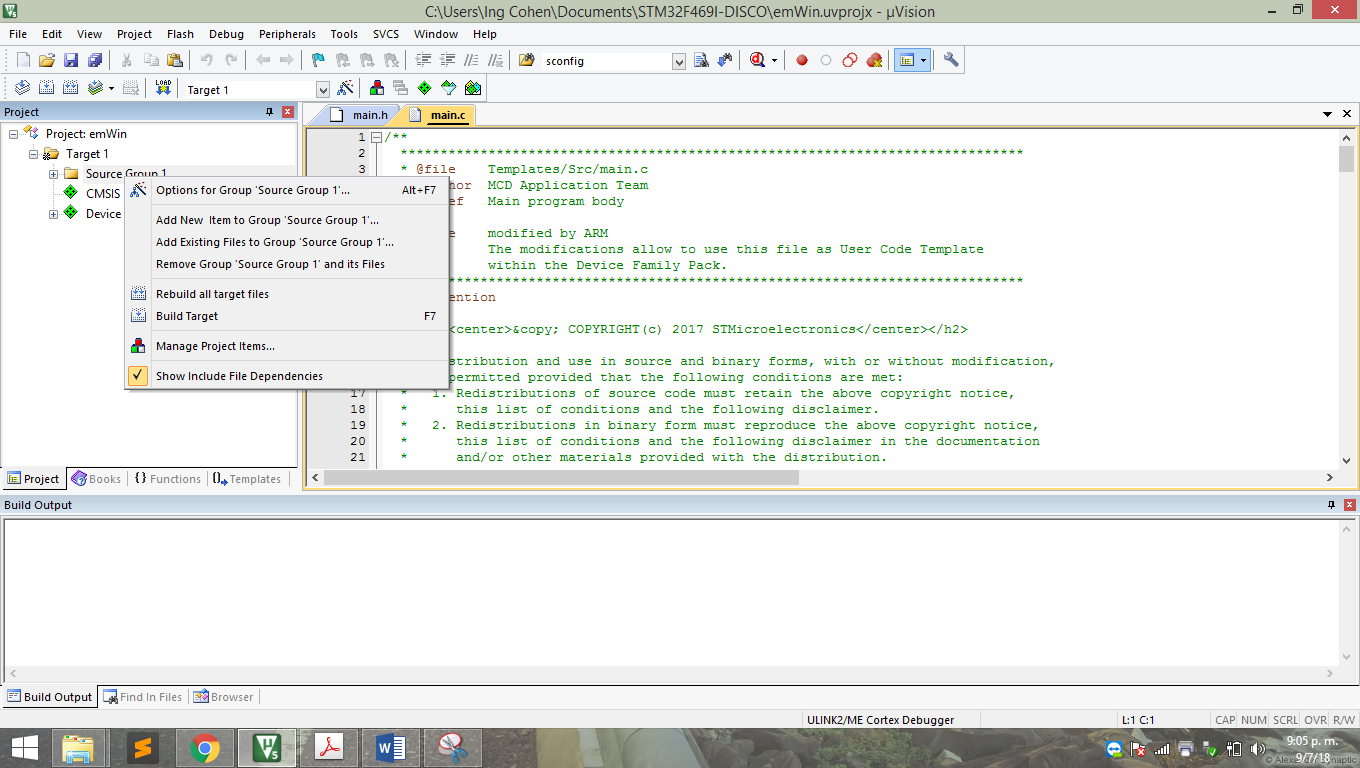
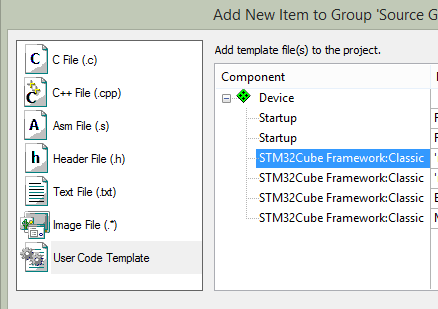
3 – Seleccione el microcontrolador STM32F469NIHx

4 – Seleccione los siguientes módulos como se muestra a continuación



5 – Dele click a resolve para terminar de resolver las dependencias y presione OK

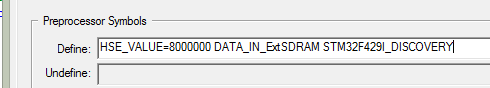
6 – Añada un nuevo módulo de las plantillas

7 – En Target Options seleccione un cristal de 12 MHz



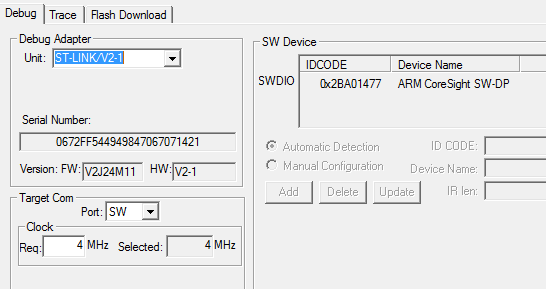
8 – En la ventana de C/C++, las directivas del precompilador serán las siguientes:

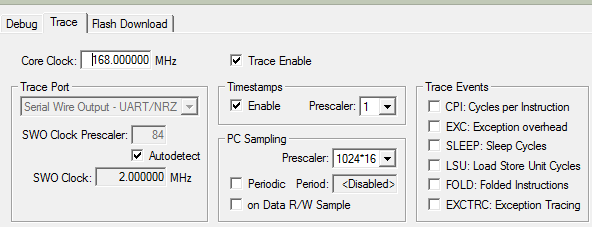


9 – Compile para verificar que no hay errores

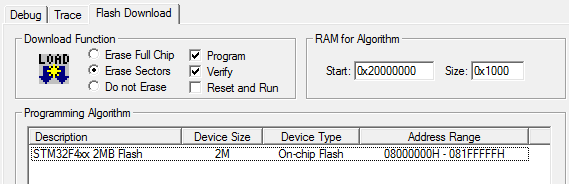
10 – Seleccione el debug adapter como ST-Link







11 – Seleccione el algoritmo de flasheo. Elimine cualquier cosa que no sea esto.



12 – Modifique la rutina de Inicialización de Reloj, aparentemente no ejecuta muy bien la inicialización a 168 MHz aún cuando se especifica el HSE\_VALUE, copie el código a continuación que si realiza la acción:

static void SystemClock\_Config(void)

{

RCC\_ClkInitTypeDef RCC\_ClkInitStruct;

RCC\_OscInitTypeDef RCC\_OscInitStruct;

RCC\_PeriphCLKInitTypeDef PeriphClkInitStruct;

/\* Enable Power Control clock \*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

/\* The voltage scaling allows optimizing the power consumption when the device is

clocked below the maximum system frequency, to update the voltage scaling value

regarding system frequency refer to product datasheet. \*/

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

/\* Enable HSE Oscillator and activate PLL with HSE as source \*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSE;

RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSE;

#if defined(USE\_STM32469I\_DISCO\_REVA)

RCC\_OscInitStruct.PLL.PLLM = 25;

#else

RCC\_OscInitStruct.PLL.PLLM = 8;

#endif /\* USE\_STM32469I\_DISCO\_REVA \*/

RCC\_OscInitStruct.PLL.PLLN = 360;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;

RCC\_OscInitStruct.PLL.PLLQ = 7;

RCC\_OscInitStruct.PLL.PLLR = 2;

if(HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

/\* Initialization Error \*/

Error\_Handler();

}

/\* Activate the OverDrive to reach the 180 MHz Frequency \*/

HAL\_PWREx\_EnableOverDrive();

/\* Select PLLSAI output as USB clock source \*/

PeriphClkInitStruct.PeriphClockSelection = RCC\_PERIPHCLK\_CK48;

PeriphClkInitStruct.Clk48ClockSelection = RCC\_CK48CLKSOURCE\_PLLSAIP;

PeriphClkInitStruct.PLLSAI.PLLSAIN = 384;

PeriphClkInitStruct.PLLSAI.PLLSAIQ = 7;

PeriphClkInitStruct.PLLSAI.PLLSAIP = RCC\_PLLSAIP\_DIV8;

if (HAL\_RCCEx\_PeriphCLKConfig(&PeriphClkInitStruct) != HAL\_OK)

{

/\* Initialization Error \*/

Error\_Handler();

}

/\* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2

clocks dividers \*/

RCC\_ClkInitStruct.ClockType = (RCC\_CLOCKTYPE\_SYSCLK | RCC\_CLOCKTYPE\_HCLK | RCC\_CLOCKTYPE\_PCLK1 | RCC\_CLOCKTYPE\_PCLK2);

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV4;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV2;

if(HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_5) != HAL\_OK)

{

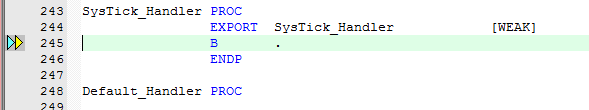
/\* Initialization Error \*/

Error\_Handler();

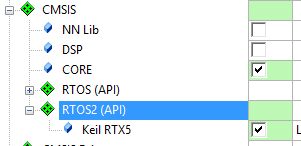
}

}

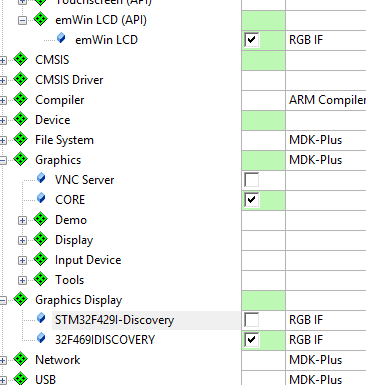
13 – Queme el programa en la memoria, ejecute el depurador y corra el programa, luego deténgalo. Este debe terminar en la función Systick Handler, la cual corregiremos más adelante añadiendo código para salir de esta interrupción (usaremos Keil RTX)



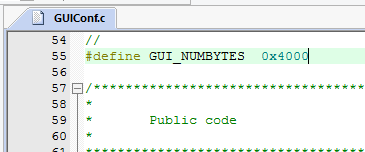
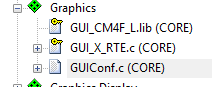
14 – Añadiremos ahora el sistema operativo



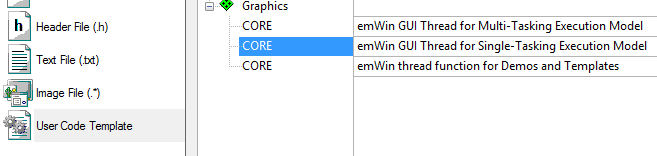
18 – Añadiremos ahora las librerías de la pantalla. Si al marcar esto salen problemas de depedencias, ponga resolver dependencias.



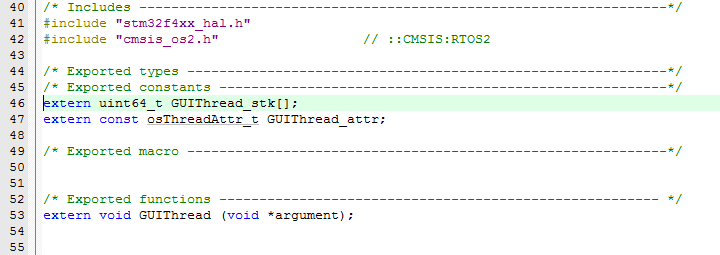
19 – Cambie esto en GUI\_Conf.c. Este es el tamaño recomendado por emWin, funciona bien para la mayoría de las aplicaciones.



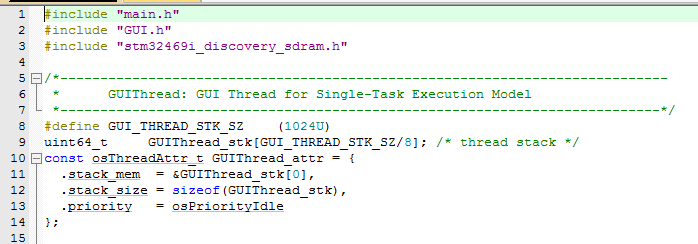
20 – Añada este archivo en su plantilla

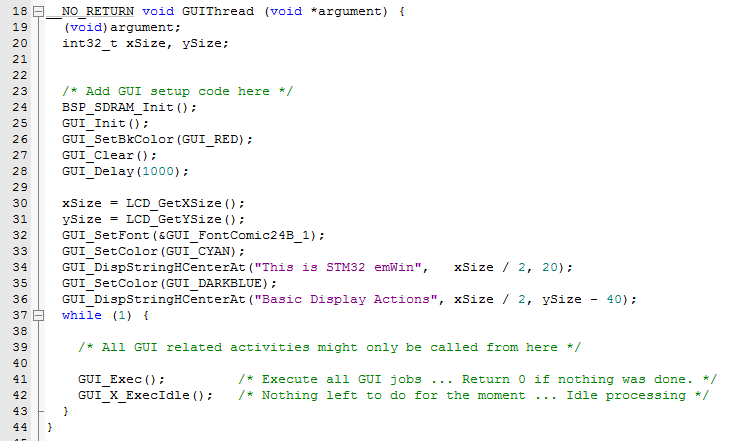


21 – Añada estas líneas a su archivo main.h

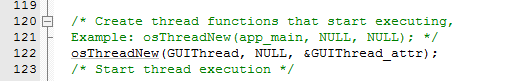


22 – Relice estas modificaciones a su archivo GUI\_SingleThread.c





23 – Cambie esto en su archivo main.c. Esto llamará la primera tarea del sistema operativo.

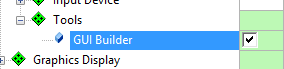


24 – Queme la aplicación y observe lo que sucede.

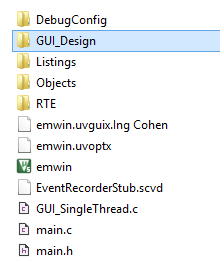
**25 – Su tarea a realizar para evaluación de esta parte del laboratorio será añadir código para dibujar un círculo en medio de la pantalla.**

Parte II – GUI Builder

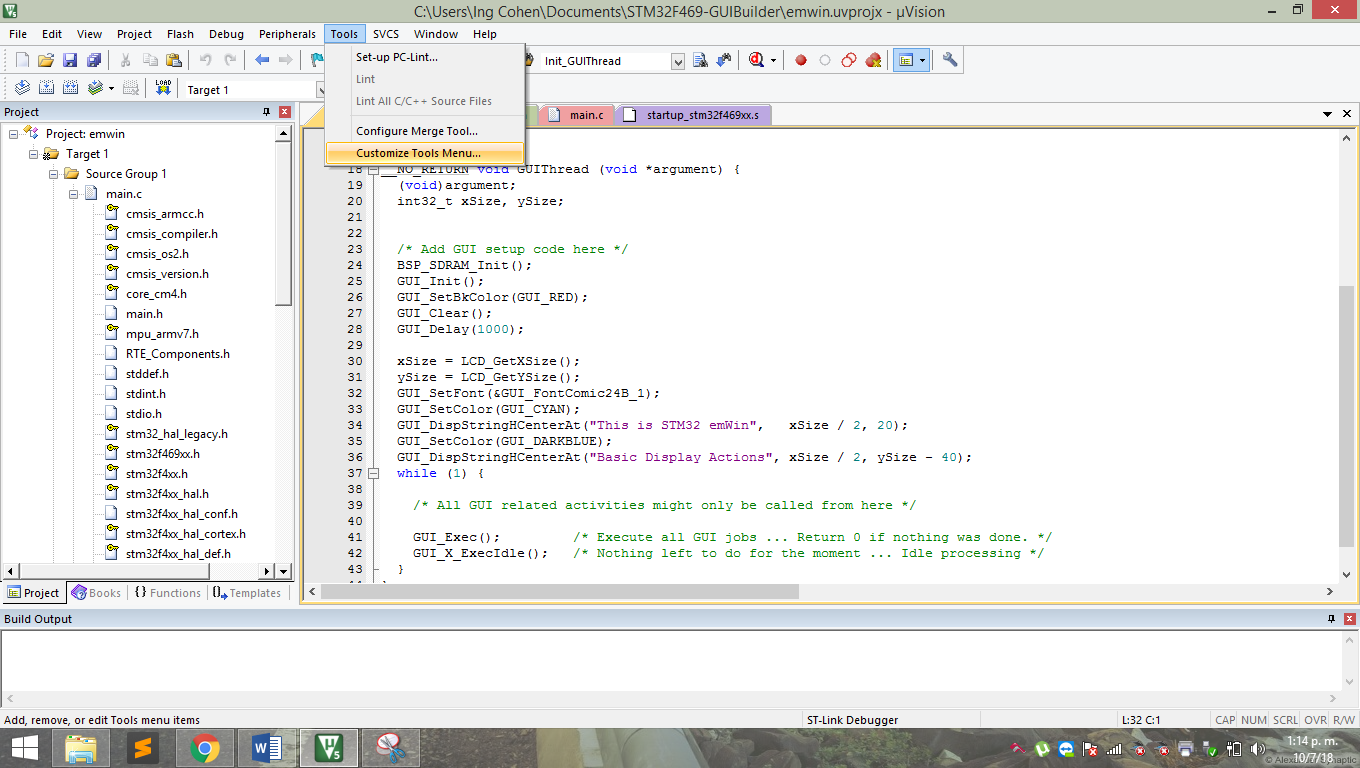
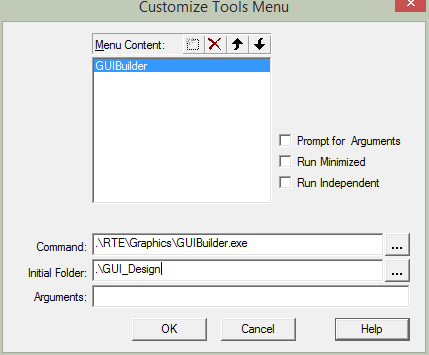
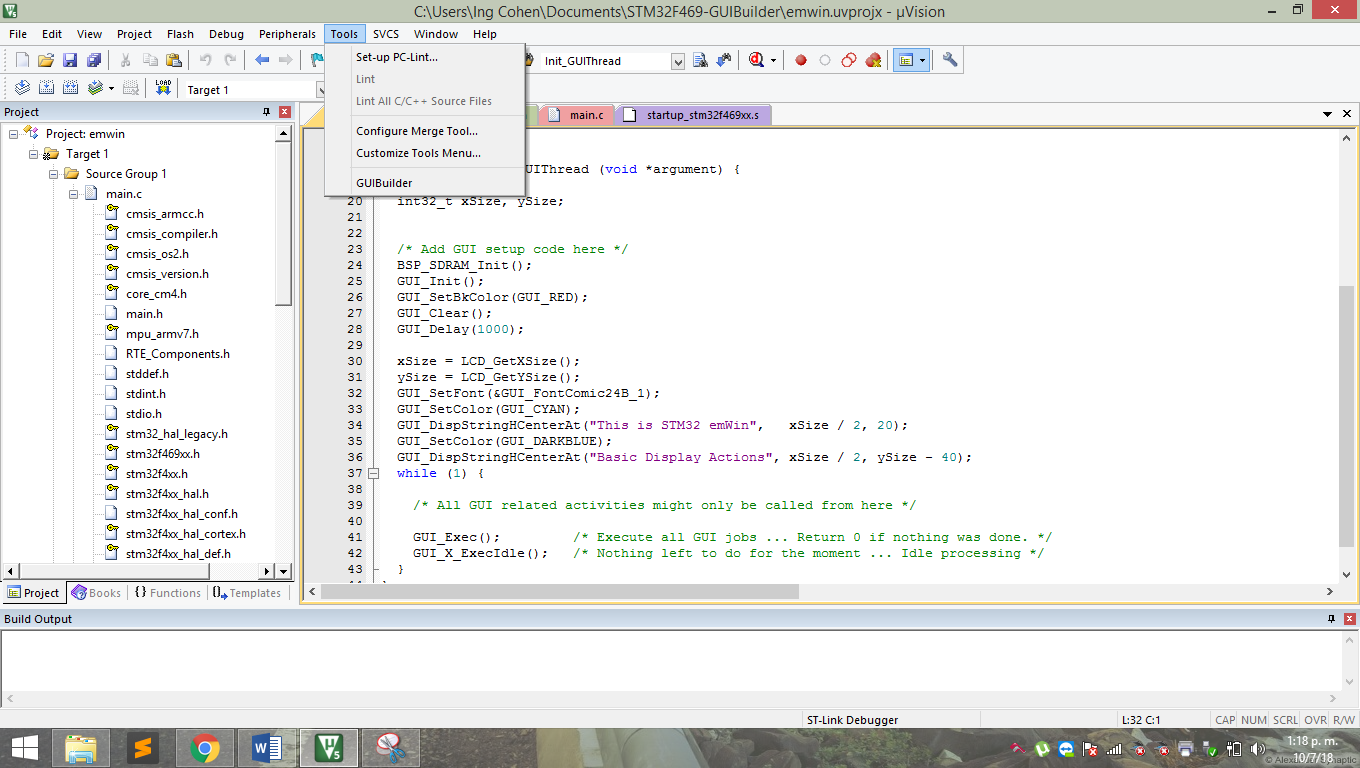
1 – Seleccione GUI Builder



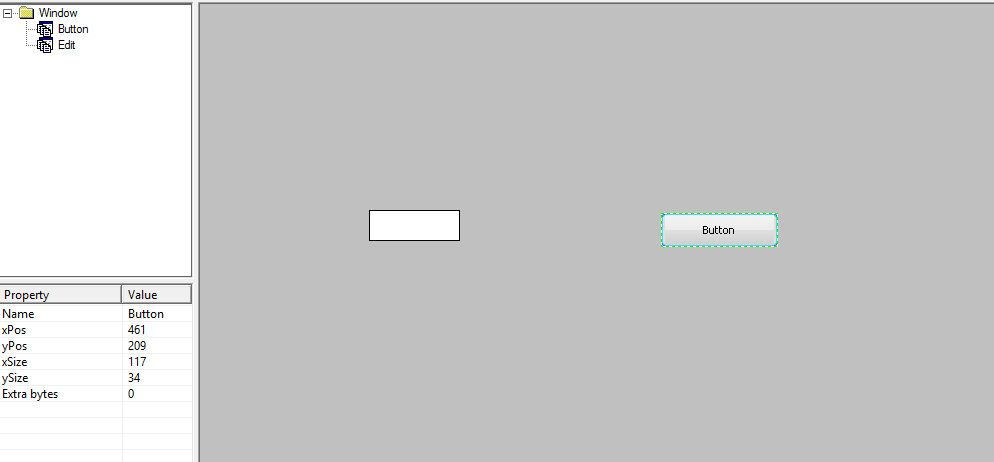
2 – Cree una carpeta para guardar su diseño, en mi caso, GUI\_Design



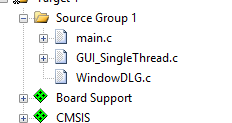
3 - Lance GUI Builder

4 – Dibuje una ventana de 800 x 480 con los siguientes elementos y salve.



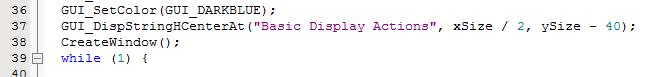
5 – Añada un archivo existente a su proyecto, recuerde que está en la carpeta raíz del proyecto dentro de la carpeta GUI\_Design



6 – Luego abra el archivo GUI\_SingleThread.c y añada las siguientes líneas



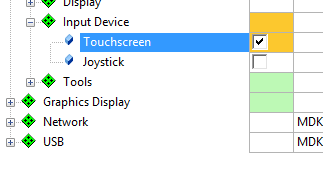




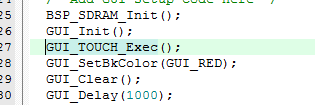
7 – Compile, queme la tarjeta y reinicie. Observe que lo diseñado es idéntico a lo desplegado.

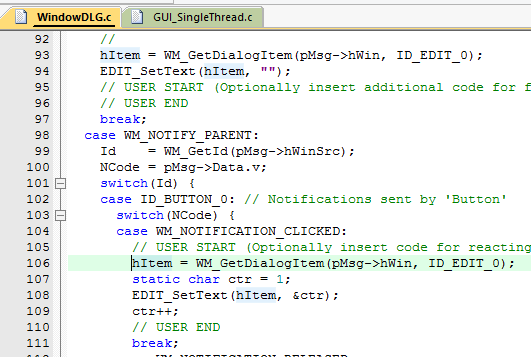
Nota: Esta sección aún no funciona, la aplicación no arranca.

8 – Seleccione el Touchscreen para aplicarlo en la pantalla. Si salen errores, resuelva las dependencias.



9 – Agruegue las siguientes líneas en GUISingleThread.c y en WindowDLG.c





10 – Compile, queme la aplicación y ejecute, observe como al oprimir el botón cambia de número.