Running TouchGFX without an operating system

TouchGFX can be configured to run without any operating system. It should be noted, however, that we recommend using an OS because it makes it *much* easier to properly utilize the MCU for other tasks without risking to interfere with the timing critical portions of TouchGFX.

GUI processing loop

The standard procedure for the GUI processing is that the GUI task calls the framework function

```
HAL::getInstance()->taskEntry();
```

which is a function that never returns. If you're not using an OS and therefore needs access to the event loop, you can replace the call to `taskEntry` with the following code, e.g. in your main function:

```
  HAL::getInstance()->enableLCDControllerInterrupt();
  HAL::getInstance()->enableInterrupts();
  while(1)
  {
      OSWrappers::waitForVSync();
      HAL::getInstance()->backPorchExited();
  }
```

Abstraction layer implementation

The TouchGFX OS abstraction layer consists of a single class, `OSWrappers`, which contains functions that are OS-specific. Normally these functions would use the semaphore mechanism of the operating system to synchronize events. They can, however, in many cases be replaced simply by code that manipulates variables. This is a complete implementation of the OSWrappers class that uses variables instead of semaphores:

```
#include <touchgfx/hal/OSWrappers.hpp>
using namespace touchgfx;

static volatile uint32_t fb_sem;
static volatile uint32_t vsync_sem;

void OSWrappers::initialize()
{
  fb_sem = 0;
  vsync_sem = 0;
}

void OSWrappers::takeFrameBufferSemaphore()
```

Recently viewed articles

TouchGFX HAL Development

Non-memory mapped external flash (e.g. NAND)

Caching bitmaps on systems with limited RAM

Lowering memory requirements using Partial Frame Buffers

Overview of TouchGFX

Related articles

Changing to a different RTOS

Non-memory mapped external flash (e.g. NAND)

Integrating CubeMX and TouchGFX

Changing to a different display

Known Issues

```
{
  while(fb_sem);
  fb_sem = 1;
}
void OSWrappers::giveFrameBufferSemaphore()
{
  fb_sem = 0;
}

void OSWrappers::tryTakeFrameBufferSemaphore()
{
  fb_sem = 1;
}

void OSWrappers::giveFrameBufferSemaphoreFromISR()
{
  fb_sem = 0;
}

void OSWrappers::signalVSync()
{
  vsync_sem = 1;
}

void OSWrappers::waitForVSync()
{
  vsync_sem = 0;
  while(!vsync_sem);
}
```

> **Note**: For a detailed description of these functions, please see this article: Changing to a different RTOS.

Of course the solution above leaves no opportunity for the processor to do other work, since it is just busy waiting on variables. In order for it to be useful, you could modify the wait loops to, for instance:

```
while(!vsync_sem)
{
    // Perform other work while waiting
    doOtherWork();
}
```

But it is important that the worker function has a short execution time, since TouchGFX relies on being notified reasonably fast when a VSYNC occurs, and similarly when the frame buffer is free. As a general rule of thumb, the maximum execution time of your `doOtherWork()` function should be:

- 1-5 millisecond when called from `waitForVSync`
- 50 microseconds when called from `takeFrameBufferSemaphore`

Longer delays than that will reduce the performance of the GUI. If you need longer execution times, or execution at predictable times, we recommend using a small RTOS like FreeRTOS which eliminates these problems entirely.

Was this article helpful? 1 out of 1 found this helpful

Have more questions? Please create a post on the forum.