STEERING WHEEL

Short textual description of the application

This project implements a steering wheel interface system based on an STM32H7 microcontroller and the ThreadX RTOS.

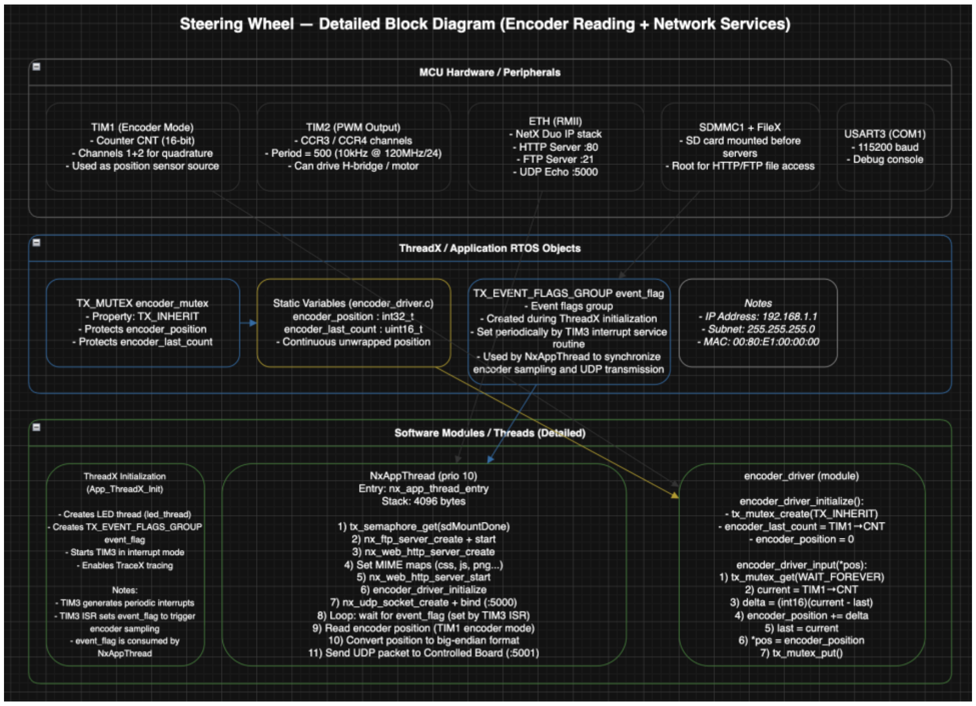
The application reads the angular position of a steering wheel through a quadrature encoder connected to TIM1 configured in encoder mode. The encoder driver implements a 16-bit wraparound algorithm to reconstruct a continuous absolute position value.

The sampling of the encoder position is driven by a hardware timer (TIM3) configured in periodic mode. At each timer expiration, an interrupt service routine sets a ThreadX event flag, which is used to trigger the acquisition of a new encoder sample within the network application thread. This approach decouples time-critical sampling from application-level processing and ensures deterministic timing behavior.

The steering wheel position is transmitted asynchronously to a remote-Controlled Board via UDP packets over Ethernet using the NetX Duo stack. This enables real-time remote control of a motor that follows the steering wheel movements.

Additionally, the system includes network services: an embedded HTTP server for serving web content from an SD card, an FTP server for file management, and a UDP-based communication channel for position transmission. The SD card is mounted using FileX before starting network services.

Application data flow


Steering Wheel — Detailed Block Diagram (Encoder Reading + Network Services)
MCU Hardware / Peripherals
TIM1 (Encoder Mode) - Counter CNT (16-bit) - Channels 1+2 for quadrature - Used as position sensor source
TIM2 (PWM Output) - CCR3 / CCR4 channels - Period = 500 (10kHz @ 120MHz/24) - Can drive H-bridge / motor
ETH (RMII) - NetX Duo IP stack - HTTP Server :80 - FTP Server :21 - UDP Echo :5000
SDMMC1 + FileX - SD card mounted before servers - Root for HTTP/FTP file access
USART3 (COM1) - 115200 baud - Debug console
ThreadX / Application RTOS Objects
TX_MUTEX encoder_mutex - Property: TX_INHERIT - Protects encoder_position - Protects encoder_last_count
Static Variables (encoder_driver.c) encoder_position : int32_t encoder_last_count : uint16_t - Continuous unwrapped position
TX_EVENT_FLAGS_GROUP event_flag - Event flags group - Created during ThreadX initialization - Set periodically by TIM3 interrupt service routine - Used by NxAppThread to synchronize encoder sampling and UDP transmission
Notes - IP Address: 192.168.1.1 - Subnet: 255.255.255.0 - MAC: 00:80:E1:00:00:00
Software Modules / Threads (Detailed)
ThreadX Initialization (App_ThreadX_Init)
- Creates LED thread (led_thread)
- Creates TX_EVENT_FLAGS_GROUP event_flag
- Starts TIM3 in interrupt mode
- Enables TraceX tracing
Notes:
- TIM3 generates periodic interrupts
- TIM3 ISR sets event_flag to trigger encoder sampling
- event_flag is consumed by NxAppThread
NxAppThread (prio 10)
Entry: nx_app_thread_entry
Stack: 4096 bytes
1) tx_semaphore_get(sdMountDone)
2) nx_ftp_server_create + start
3) nx_web_http_server_create
4) Set MIME maps (css, js, png...)
5) nx_web_http_server_start
6) encoder_driver_initialize
7) nx_udp_socket_create + bind (:5000)
8) Loop: wait for event_flag (set by TIM3 ISR)
9) Read encoder position (TIM1 encoder mode)
10) Convert position to big-endian format
11) Send UDP packet to Controlled Board (:5001)
encoder_driver (module)
encoder_driver_initialize():
- tx_mutex_create(TX_INHERIT)
- encoder_last_count = TIM1→CNT
- encoder_position = 0
encoder_driver_input(*pos):
1) tx_mutex_get(WAIT_FOREVER)
2) current = TIM1→CNT
3) delta = (int16)(current - last)
4) encoder_position += delta
5) last = current
6) *pos = encoder_position
7) tx_mutex_put()

The data flow of the application is organized around the encoder sampling, timing, and network transmission components.

The quadrature encoder connected to the steering wheel generates two phase-shifted signals (A and B) that are fed into TIM1 configured in encoder mode. The timer hardware automatically increments or decrements its counter based on the rotation direction, providing a 16-bit position value.

A separate hardware timer (TIM3) is configured to generate periodic interrupts at a fixed sampling rate. Each TIM3 interrupt invokes a callback function that sets a ThreadX event flag. This event flag acts as a synchronization mechanism between the interrupt context and the network application thread.

The encoder driver module reads the TIM1 counter register when the event flag is received and performs wrap-around handling using signed 16-bit arithmetic. This technique allows reconstructing a continuous 32-bit absolute position even when the 16-bit counter overflows. Access to the encoder state variables is protected by a mutex to ensure thread-safe operation.

A network application thread (NxAppThread) runs within the NetX Duo stack. This thread waits for the event flag generated by the timer interrupt, reads the encoder position, converts it to bigendian format and transmits the sampled position via UDP to the remote-Controlled Board.

The HTTP server handles incoming requests through a callback function that redirects root requests to index.html and can process custom endpoints. The FTP server provides file access to the SD card with open authentication.

A separate link check thread monitors the Ethernet connection status periodically and re-enables the network driver when the link is restored after disconnection.

Description of Global Functions and Variables

This section describes the main global variables and application-specific functions implemented in the project, including their purpose, parameters, and return values.

Global Variables and RTOS Objects

encoder_position (int32_t)
Static variable representing the absolute unwrapped encoder position in ticks.
It is updated by the encoder driver upon each read and provides a continuous position value that handles counter wrap-around. Access is protected by encoder_mutex.

encoder_last_count (uint16_t)
Static variable storing the previous TIM1 counter value.
Used to compute the position delta between consecutive reads. Protected by encoder_mutex.

encoder_mutex (TX_MUTEX)
Mutex used to protect access to encoder state variables.
Created with TX_INHERIT property to prevent priority inversion. Acquired by any thread reading the encoder position.

event_flag (TX_EVENT_FLAGS_GROUP)
Event flags group used to synchronize the periodic timer interrupt with the network application thread.
The flag is set inside the TIM3 interrupt service routine and consumed by the network thread to trigger encoder sampling and UDP transmission.

Encoder Driver

encoder_driver_initialize(void)

Initializes the encoder driver by creating a mutex and resetting internal state variables.
• Description:
Creates encoder_mutex with TX_INHERIT property, reads initial TIM1->CNT value, and sets encoder_position to zero.
• Return value: TX_SUCCESS on success or a ThreadX error code otherwise.

encoder_driver_input(int32_t *position)

Reads the current encoder position with wrap-around handling.
• Parameter: pointer to an int32_t variable where the absolute position is stored.
• Description:
The function acquires the mutex, reads TIM1->CNT, computes delta using signed 16-bit subtraction (which handles wrap-around automatically), updates the internal position, and releases the mutex.
• Return value: TX_SUCCESS on success, ThreadX error code on failure.

Network Application Thread

nx_app_thread_entry(ULONG thread_input)

Main NetX Duo application thread implementing network services.
• Parameter: thread_input (unused ThreadX initialization parameter).
• Description:
Waits for the SD card mount completion semaphore, then creates and starts an FTP server and an HTTP server with MIME type mappings. Initializes the encoder driver, creates and binds a UDP socket on port 5000 and enters an infinite loop waiting for an event flag set by a timer interrupt. Upon event notification, it reads the encoder position, converts it to big-endian format, and transmits the position data via UDP to a remote board at a predefined IP address and port.
• Return value: None (infinite thread loop).

App_ThreadX_Init(VOID *memory_ptr)

Main ThreadX application initialization function.
• Parameter: memory_ptr (pointer to the ThreadX byte pool used for dynamic allocations).
• Description:
Allocates stack memory and creates an auxiliary LED thread. Creates a ThreadX event flags group used for synchronization between timer interrupt context and application threads. Starts a hardware timer (TIM3) in interrupt mode to generate periodic events for encoder sampling. Enables TraceX tracing for runtime analysis and debugging.

• Return value: TX_SUCCESS on successful initialization, or a ThreadX error code on failure.