Project Overview

Table of contents

Development Environment

STM32 / C++:

IDE	-	Atollic Truestudio v9.3
HAL generation	-	CubeMX v6.0.1
MCU	-	STM32F042K6
Dev Board	-	NUCLEO-F042K6

PC / Python:

IDE	-	Visual Studio Code v1.54.3 (Windows)
Python	-	Python v3.9

Logic Summary

Note:
***Flowcharts** can be found in the Documentation folder generated using www.draw.io as both .pdf files.*
***Videos** of the system have been uploaded to a private playlist on YouTube: https://youtube.com/playlist?list=PLDWfgviIgeVMS9EBJ9NjDTaawb84WkjV9*

My implementation of the task involves the use an STM32F0 as the target device. The PC is the master that uses a python-based GUI to transmit instructions through a USB to Serial converter.

Once started, the python application will allow the user to open a com port from the list of available com ports. To begin an operation, the user types a number from 1 to 100 on the terminal on the PC. This value is encoded into a transmit message 10 bytes long. The number is placed in the instruction slot (byte 0). A crc16 is generated using this message and appended to the end. Final transmission of data to the target device is 12 bytes long. Transaction is complete when an acknowledge is received from the target. If a not-acknowledge is returned, transmission is considered failed. The data sent is echoed on the terminal only if an acknowledge was received.

If the user attempts to send several instructions, they must wait for an instruction to be transmitted and acknowledge/not acknowledge received from the target before the next can be sent.

The target device uses a timer to track when 10 seconds have passed since an instruction went into effect and a pwm to generate the led binking.

When the target device receives an instruction message from the pc, it stores it in a receive buffer of 12 bytes. A crc16 calculation is performed on received data to check for any data loss. If succeeded, the instruction byte is placed into an 10 byte serial buffer for implementation and an acknowledge is returned, else a not-acknowledge.

The serial buffer is a ring buffer with indexes that track which data has been processed, which is pending and how many are pending. Once an instruction is placed in the buffer, the device can take one of two actions:

a) If this is the first instruction or this instruction is coming more than 10 seconds since the transmission of the previous instruction, the system must set the pwm register and enable the timer and pwm.
b) If this instruction arrives less than 10 seconds since the previous instruction, the system only needs to update the pwm register and restart the timer once the previous instruction has completed.

When an instruction is going into effect, apart from setting the timer and pwm, the device also performs a check on the instruction. It checks if the instruction value is a multiple of 4 and/or 7. Depending on the result, it prepares a response message to be returned to the PC. This message can range from 1 to 10 bytes. A crc16 is calculated and appended to the end of the message. Final transmission size is always 12 bytes.

If more than 10 instructions are filled in the buffer, all additional instructions are ignored and a not-acknowledge returned until the first instruction operation is complete and a slot is cleared to be written to. This is to prevent a buffer over-write of previous instructions.

If there are no more instructions waiting in the serial buffer queue, the device disables the pwm and timers upon competition of the final instruction. No actions are taken until a new instruction is received.

Project Breakdown

STM32F042K6

a) Higher Application
    a. blink.c : Enables/Disables LED, generates and transmits response message
    b. multiples.c : Checks for multiple of 4 and/or 7 and builds response message
    c. definitions.h : Definitions for buffer sizes, ring buffer structure, return codes etc
b) Middleware
    a. serial_port.c : UART transmit/receive, integrity check, DMA transfer data to buffer
    b. timer.c : Enable/disable timer and pwm peripherals, register timer callback
    c. crc.c : Perform crc16 check, generate crc16, append crc to buffer
    d. dma.c : Register DMA success/failure callbacks

PC

a) UI.py : Serial Terminal GUI

Critical Design Considerations

1. **Hardware based CRC16:**
   Several MCU's come with hardware CRC peripherals which are much faster. The STM32L4 does not but M4 does. The STM32F042K6 used here has a CRC32 peripherals, not CRC16. Hence, chose to implement a software CRC16. Code will function the same when porting to another STM with/without CRC peripherals.
2. **CRC16 Lookup Table:**
   A CRC algorithm can be implemented faster by using a lookup table. Instead of calculating the modulo 2 division for each byte, a lookup table is used to improve processing time. However, it takes more space in memory.
3. **Automatic Baudrate Selection:**
   The STM32F0 and L4 have an ABR option for some UART peripherals. This would make the system more adaptable for different serial configurations. M4 does not, hence chosen to remove feature after implementation.
4. **FIFO Ring Buffer:**
   A ring buffer was chosen to store valid instructions as it allows for greater freedom than a linear buffer. Data can be written to slots as soon as they are processed.
   Buffer is static at 10 bytes. Chose not to implement a dynamic buffer as over time, constantly reserving and freeing memory on the heap can corrupt it.

Known Issues

1. When sending the first instruction through the GUI resulting in an immediate return of the response message, the python code sometimes misses the first response from the STM. I have confirmed the STM always sends it immediately once the instruction is received. Unsure why this bug is occurring.
   Possible reason: The time between the serial_read thread can take control of the COM port after *serial_transmit* sends data maybe too long.

Final Notes

1. All file headers contain feature lists stating detailed features and limitations of the library.
2. All file headers are doxygen compliant.
3. All references have been linked in the readme file.