Testy Maintainer

Yuchen Zhou

UID: 115243248

12/16/2020

Table of Contents

Abstract

Testy is a software that runs on a STM32 NUCLEO-H745ZI-Q microprocessor board. Testy contains a set of devices that are controlled by plain-text messages sent over to the board using the microprocessor's serial debug port from the PC. These plain-text messages are designed commands with specific parameters. Actions in the Testy command will be interpreted by the software CPU to dispatch different functions on the H745 microprocessor. The board will input or output analog or digital values and record or communicate data. Testy report status and results back to the PC over the serial debug port when requested or on an on-going depending on the specific device function selected.

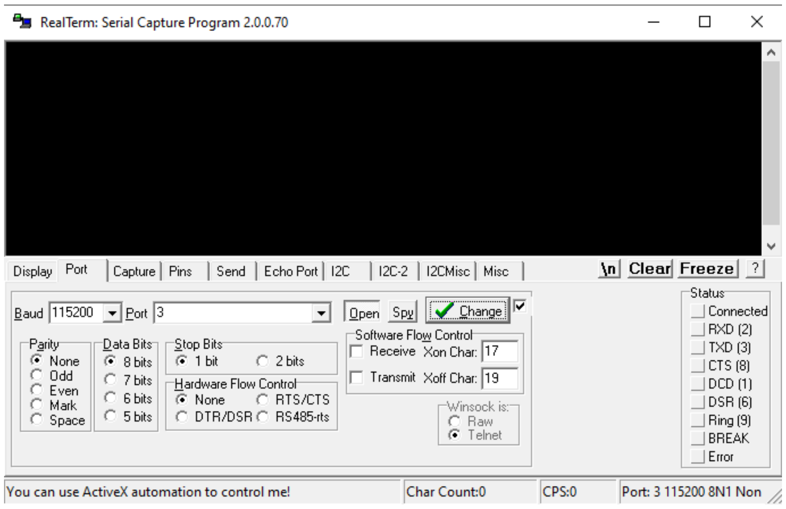
For the specific devices, Testy can control following four devices with various functionalities: GPIO, TIMER, DAC, ADC. Users can establish serial communication with the board using Realterm by properly select the baud and communication port, which Realterm is free Serial/TCP Terminal that can be used to send ASCII commands using serial ports.

***Note: Referring to the code will be helpful to understand the functions while reading this document.*

User Interface

Before sending detailed commands to the microprocessor board, users must first compile and load the compiled binary files onto the board. System Workbench for STM32, a free IDE developed by ST, is recommended for code compilation, and debugging the software step by step. Once the software is uploaded onto the board, user now can open and setup Realterm for serial communication.

First, user should select the proper communication baud and port number. Baud is always 115200, but port number is depended on the PC. Check PC's device manager if multiple serial devices are connected to the PC.



Baud 115200 Port 3

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

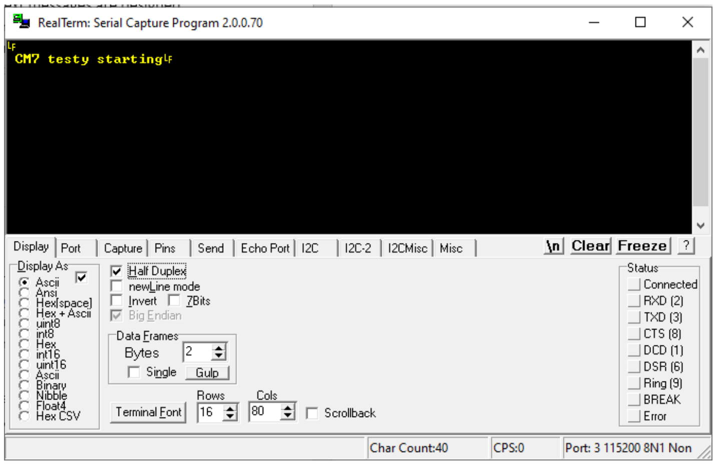
Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19

Winsock is: Raw [✓ Telnet]

You can use ActiveX automation to control me! | Char Count:0 | CPS:0 | Port: 3 115200 8N1 Non

Figure 1. Realterm Port Setup

Next, user can go to the "Display" tab. Select "Half Duplex" to display ASCII characters. Once the user has completed previous two steps, the user may press the reset button (black push button) on the NUCLEO board to reset and start the program. Following screen should show up after pressing the reset button:



RealTerm: Serial Capture Program 2.0.0.70

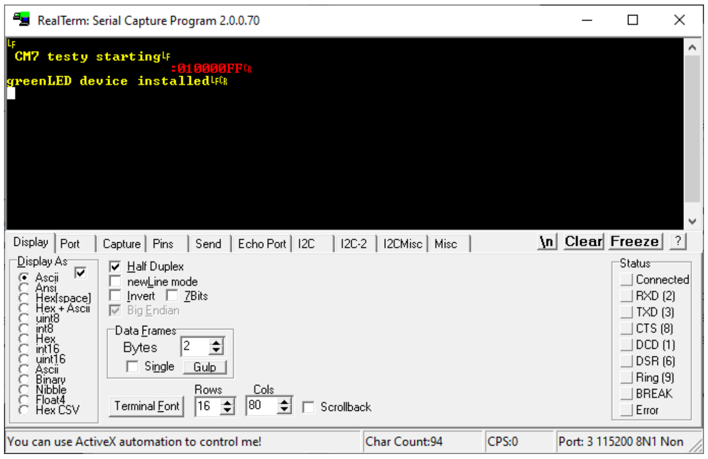
```
LF
 CM7 testy startingLF
```

Display As: Ascii [✓] · [✓] Half Duplex · [ ] newLine mode · [ ] Invert · [ ] 7Bits · [✓] Big Endian · Data Frames: Bytes 2 · [ ] Single · Rows 16 · Cols 80 · [ ] Scrollback

Char Count:40 | CPS:0 | Port: 3 115200 8N1 Non

Figure 2. Testy is ready to use

Now, before entering any commands, make sure read the "Reminder" section of this document, there are some prerequisite device initializations that must be called or wirings for some device functionalities. Once that is done that, the user may enter command (for specific command structures, please refer to the user's manual) in ASCII characters using the Realterm display window. All valid commands should begin will the ":" character. For feedback, an ASCII string that reports the status of the command user entered will be returned in the Realterm display window, if the input command is valid. Otherwise, nothing is going to display after the user presses "Enter".



RealTerm: Serial Capture Program 2.0.0.70

```
LF
 CM7 testy startingLF
                  :010000FFCR
greenLED device installedLFCR
```

Display As: Ascii [✓] · [✓] Half Duplex · [ ] newLine mode · [ ] Invert · [ ] 7Bits · [✓] Big Endian · Data Frames: Bytes 2 · [ ] Single · Rows 16 · Cols 80 · [ ] Scrollback

You can use ActiveX automation to control me! | Char Count:94 | CPS:0 | Port: 3 115200 8N1 Non

Figure 3. Valid Command with Feedback

Note: User may always press the reset button to restart Testy

Testy Control Flow

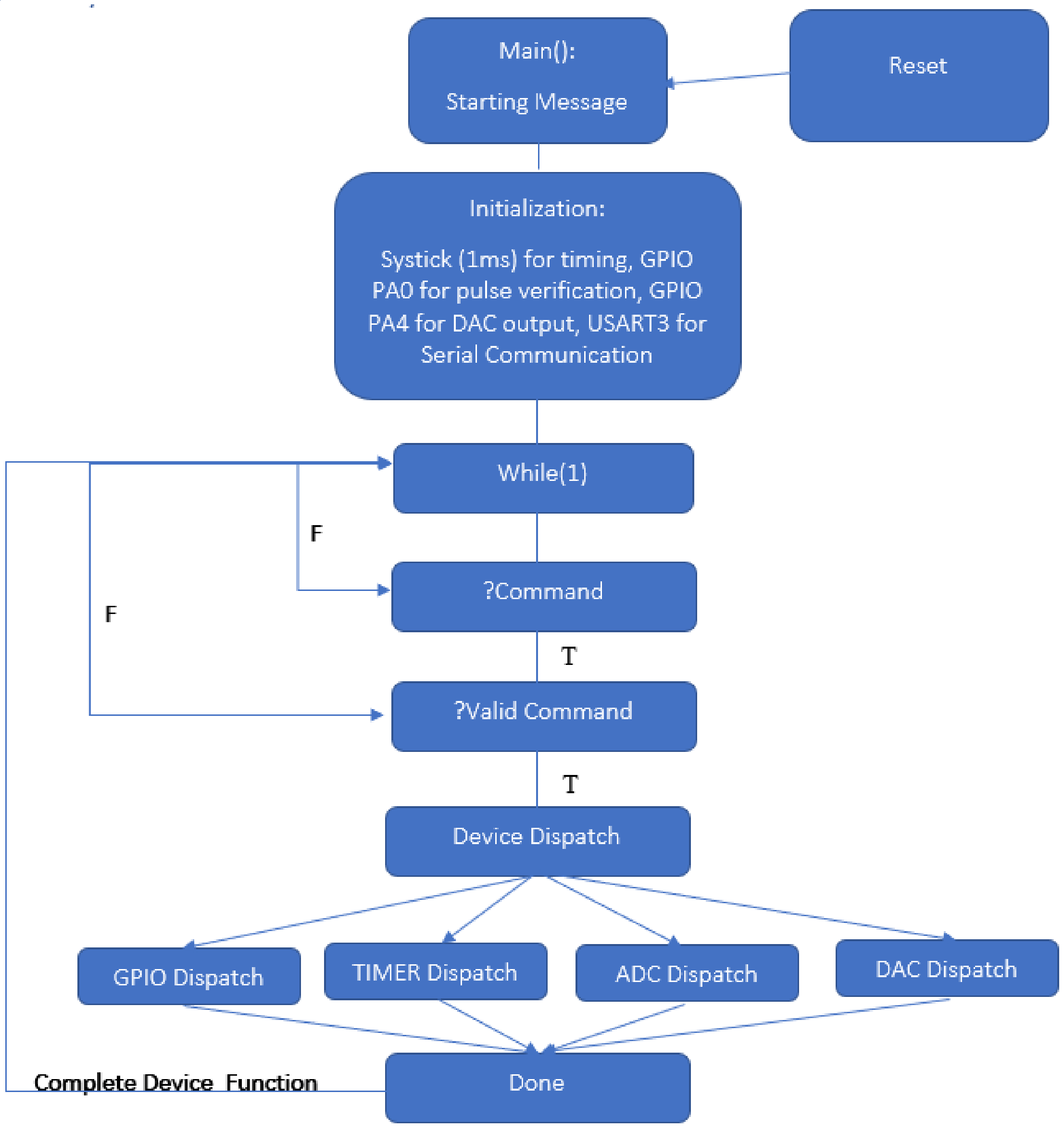


Figure 4. Testy Control Flow

***Note: Each device has its own dispatch table to activate the device function*

Initializer Functions

initSysTick64MHz

This function initializes the Systick Timer at 64MHz and the Systick Interrupt Handler at 1000Hz. Many device functions will use the Systick handler as the timer source

USART3_HWinit

This function initializes the USART chip to communicate at a baud rate of 115200bits/sec with no parity bit, 8 data bits, and 1 stop bits.

GPIO_A_Config

This function initializes GPIO pin PA0 to input, allowing it to be used for signal verification. As a side effect, this function also activates GPIOA in the RCC.

GPIOA4_init

This function initializes GPIO pin PA4 to analog, allowing it to be used for DAC output. As a side effect, this function also activates GPIOA in the RCC.

Hexcmd2bin

This function converts ASCII hex characters into binary numbers. This function also verifies whether the input command is acceptable.

*****Note: for the exact command structure and function numbers of all devices, please refer to the user's manual***

GPIO Capability

Immediate functions

Init_Green_LED: this function initializes the GPIO Pin PB0 that is connected to the green LED as output.

Set_Green_LED: this function turns the green LED on.

Reset_Green_LED: this function turns the green LED off.

Report_Green_LED_status: this function reports the status of the green LED on/off

Stop_Green_LED_blink: this function stops the green LED from blinking

Toggle_Green_LED: this function toggles the current state of the green LED

Init_Push_Button: this function initializes the GPIO Pin PC13 that is connected to the blue push button on the NUCLEO as input. The user can use the push button to control the LED

Stop_Push_Button: this function turns off the push button off

Toggle_via_push: this function allows the user to press the push button <u>once</u> to toggle the Green

Scheduled functions

All functions in both immediate and repetitive sections can be scheduled for late activation (between 0 and $2^{16} - 1$ milliseconds), please refer to the user's manual about how to schedule functions for late activation

Repetitive functions

Slow_blink_Green_LED: this function blinks the green LED at 1Hz

Fast_blink_Green_LED: this function blinks the green LED at 5Hz

Toggle_via_push: this function allows the user to press the push button to toggle the Green （same function as **Toggle_via_push** described in the “immediate section”, the user have to use the command parameter to select immediate/repetitive）

TIMER Capability

Init_TIM1: this function configures timer1, making the counter to count every microsecond. This function also configures GPIO PE11 as an alternate function pin for tim1 output.

TIM1_PWM**:** this function takes two parameters, duty cycle in percentage and period in microseconds. TIM1 will output a Pulse Width Modulation signal using PE11 based on the duty cycle and period

TIM1_PFM**:** this function takes two parameters, active high duration, and period in microseconds. TIM1 will output a Pulse Frequency Modulation signal using PE11 based on the pulse width and period

TIM1_Pulse**:** this function takes two parameters, active high duration, and period in microseconds. TIM1 will output a single Pulse Frequency signal using PE11 based on the pulse width and period

TIM1_Pulse_Width**:** this function takes two parameters, duty cycle in percentage and period in microseconds. TIM1 will output a single Pulse Width signal using PE11 based on the duty cycle and period

TIM1_Pulse_Frequency**:** this function takes two parameters, active high duration, and period in microseconds. TIM1 will output a single Pulse Frequency signal using PE11 based on the pulse width and period

TIM1_Measure_Pulse**:** This functions measures and reports pulse frequency and active high duration of the signal generated by TIM1. *Make sure use a jumper to connect PE11 and PA0, so PA0 can measure the pulse width and period.*

TIM1_Stop_Pulse**:** this function will stop current pulse that is generated by TIM1

DAC Capability

DAC_Init**:** This function enables DAC channel 1 for analog signal output

DAC_constant_supply**:** this function takes a parameter between 0-4095, making the DAC to output a constant voltage supply between 0 and 3.3 at pin PA4 based on the following equation:

$$Vout = \frac{parmeter}{4095} * 3.3V$$

DAC_PWM_50**:** this function takes two parameters, active high duration in milliseconds and voltage level in digital representation (0 – 4095), generating a PWM signal with duty cycle = 50% , active high duration = duration parameter, and amplitude = $Vout$ above

DAC_SINE**:** This function takes one parameter period in millisecond (0 to $2^{16} - 1$), generating a sine wave of that period with amplitude = 1.65V centered at 1.65V

DAC_Stop_Signal**:** this function stops the DAC from generating signals

ADC Capability

ADC_Input_init: This function initializes GPIO pin PC2 as analog for ADC3 input and ADC3 for analog signal conversion.

Start_ADC3_EOC_IRQ: This function enables the ADC3's interrupt handler when one conversion of data is completed. Using ADC3_IRQHandler to store converted analog data (0 to $2^{16} - 1$) into memory. *Please make sure connect the input signal to GPIO pin PC2.*

ADC_Start_Conversion: this function makes ADC3 converting analog input to pin PC2. ADC3_IRQHandler will not fire until this function is called.

Systick_IRQ_Handler

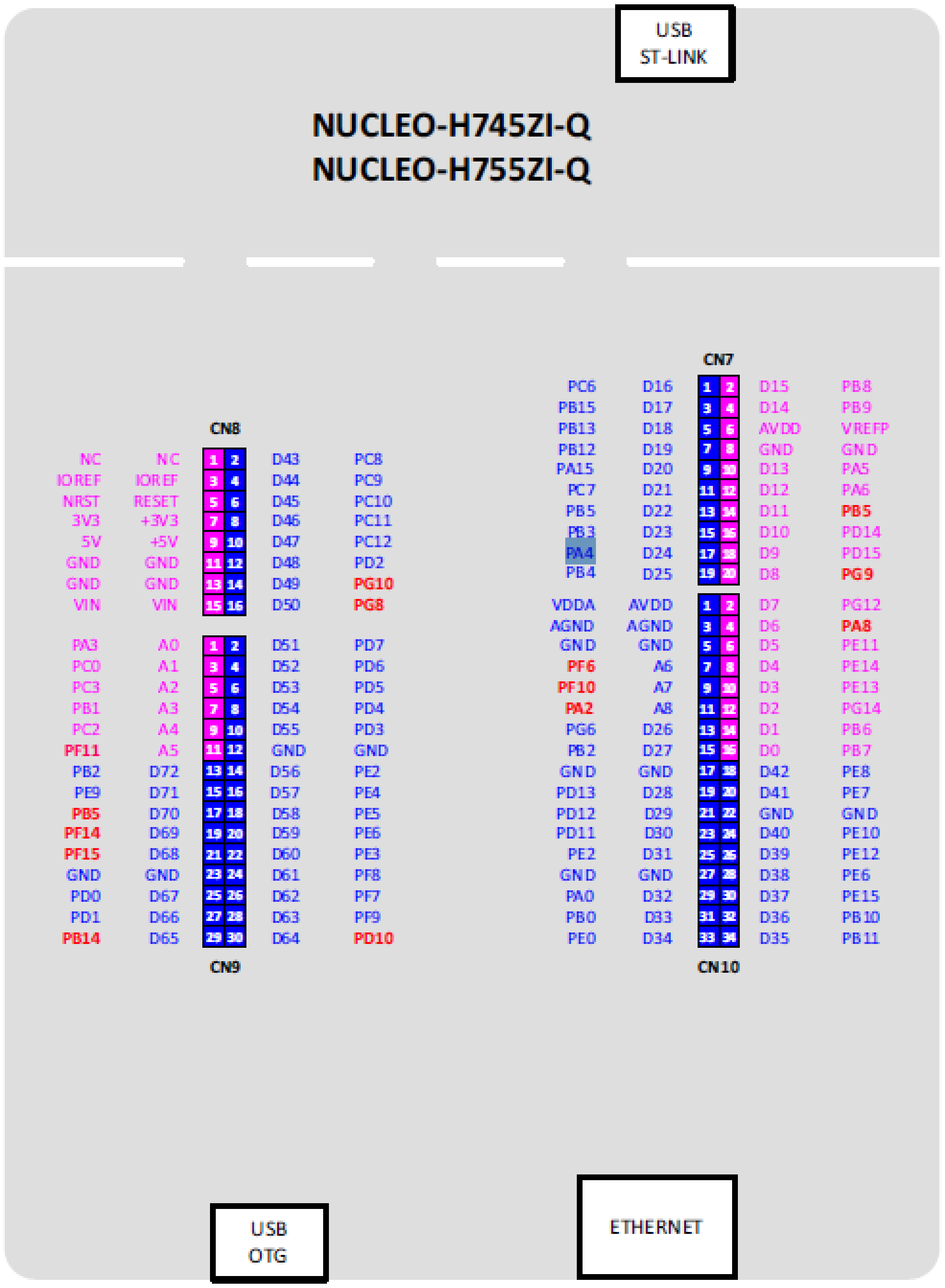
The Systick interrupt handler embodies following features:

1. Schedule delayed GPIO functions
2. Toggle the LED state for blinking
3. Push button debouncing implementation. The LED will not be triggered unless 3 milliseconds of low followed by high
4. DAC PWM and sine waves counter

Reminder/Limitations

1. For the GPIO device, make sure initialize the green led and the blue push button before using other functions
2. For the GPIO device, do not enter a new schedule function until previous one has completed. Otherwise, the previous tasks will be over written.
3. For the Timer device, make sure use a jumper to connect PE11 output and PA0, so PA0 can measure the pulse width and period. Also, make there is a real PWM signal going on, otherwise the measure pulse function will never return.
4. For the ADC device, make sure connect the analog signal that the user wants to convert to pin PC2, otherwise the interrupt handler will never fire
5. Make sure call the initialization functions before using the devices to output/input.
6. !! Don't connect a voltage source that is higher than 3.3V directly to the pins, that will damage at least the pin, worst case the board.

Nucleo Board Pin Map (For Reference)



PXX : differences compared with NUCLEO-H743ZI

🟪	Arduino subset of Zio =	A0 to A5 and D0 to D15
🟦	Zio extension =	A6 to A8 and D16 to D72