Testy User Manual

Yuchen Zhou

UID: 115243248

12/16/20

Contents

Device/Function/Function Parameters Information

Beside the reset function, all valid commands are written in the following format:

‘:’ + byte for device # + byte for function # + parameters if applicable + compensator

Compensator is a value that makes the least significant **byte** of the sum of all bytes of the command equal to 0. For example, “:010000FF” is a valid input command. 0xFF is the compensator to 0x010000 as 0x01 + 0xFF in hex is 0x100 = 0b100000000, which the least significant byte is 0. For user convenience, the user can **hex2Testy.exe** that is attached to the .zip to create Testy-acceptable commands.

In the following sections, device numbers, function numbers, and functions parameter and acceptable range will be listed

Device Numbers

Device Number	Device Name
1	GPIO
2	TIM
3	ADC
4	DAC

Table 1. Testy Device Numbers

Function Numbers

Function Number	GPIO	TIM	ADC	DAC
0	Inti_Green_LED	Init_TIM1	ADC_Input_init	DAC_Init
1	Set_Green_LED	TIM1_PWM	Start_ADC3_EOC_IRQ	DAC_constant_supply
2	Reset_Green_LED	TIM1_PFM	ADC_Start_Conversion	DAC_PWM_50
3	Report_Green_LED_status:	TIM1_Pulse		DAC_Stop_Signal
4	Slow_blink_Green_LED	TIM1_Pulse_Width		DAC_SINE
5	Fast_blink_Green_LED	TIM1_Pulse_Frequency		
6	Stop_Green_LED_blink	TIM1_Measure_Pulse		
7	Toggle_Green_LED	TIM1_Stop_Pulse		
8	Init_Push_Button			
9	Toggle_via_push			
A	Stop_Push_Button			

Table 2. Testy Functions Numbers

Function Parameters

GPIO

Parameter Name	Description	Applied to
Scheduled (1) (required)	00: immediate execution 01: delayed If 01, the next two bytes will be delayed time from 0x0000 - 0xffff in ms If 00, no need for extra parameters	All GPIO functions
Immediate/repetitive (2) (required)	00: immediate 01: repetitive If 00, the push button only works for once If 01, the use of the push button is unlimited	Toggle_via_push

Table 3. GPIO function parameters

***Note: () indicates parameter order*

ADC

N/A

TIM

Parameter Name	Description	Applied to

Period (1)	0x0000 – 0xffff in us	All but {
		Init_TIM1
		TIM1_Measure_Pulse
		TIM1_Stop_Pulse
		}

Duty circle (2)	0x0000 – 0x0064 in %	TIM1_PWM TIM1_Pulse_Width
Active High Duration (2)	0x0000 – 0xffff in us Must be less than period	TIM1_PFM TIM1_Pulse TIM1_Pulse_Frequency

Table 4. TIM function parameters

DAC

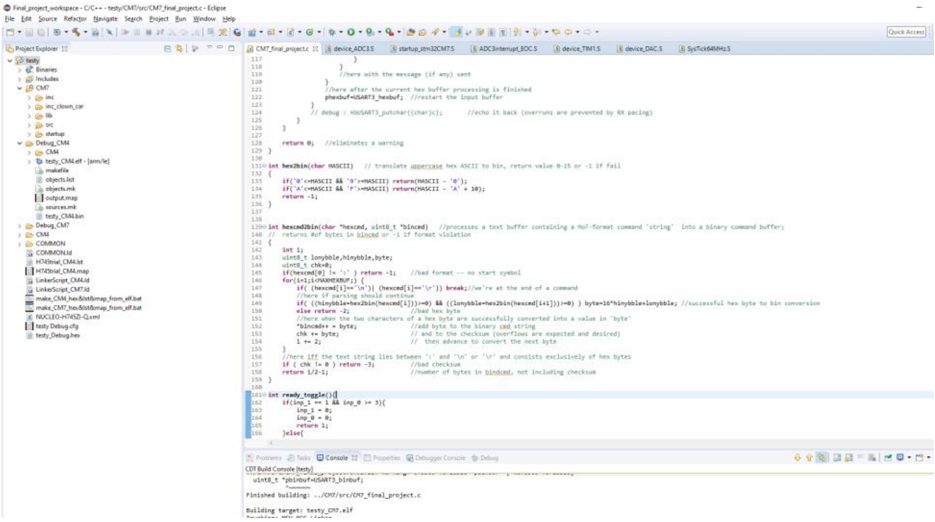
Parameter Name	Description	Applied to
Voltage Level (1)	0x0000-0x0fff	DAC_constant_supply DAC_PWM_50
Period (1)	0x0000-0xffff in ms	DAC_SINE

Table 5. DAC function parameters

***Note: look at the test scripts for example commands if you don’t fully understand the structure of the commands*

Running the program

1. Download Realterm and System Workbench for STM32 software mentioned in the “User Interface” section of the maintainer document.
2. Open System Workbench for STM32, select the “Final_project_workspace” as the workspace path. Open it.
3. Select “Import existing projects” if the project window is not really opened. Select “Final_project_workspace” as the path. Open Testy Project.

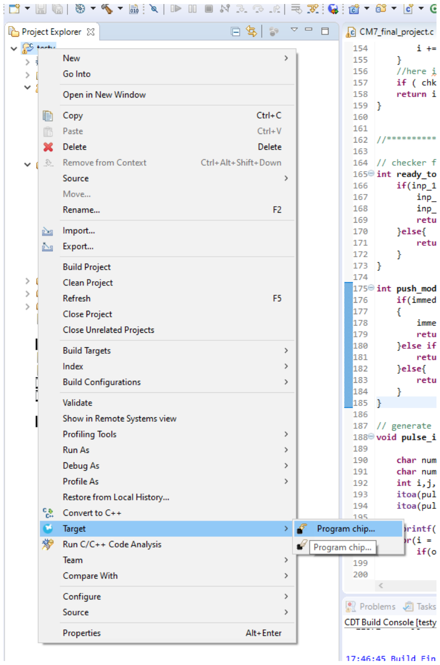


```
                    }
                }
                //here with the message (if any) sent
            }
            //here after the current hex buffer processing is finished
            phexbuf=USART3_hexbuf;  //restart the input buffer
        }
    // debug : nbUSART3_putchar((char)c);      //echo it back (overruns are prevented by RX pacing)
        }
    }

    return 0;   //eliminates a warning
}

int hex2bin(char HASCII)   // translate uppercase hex ASCII to bin, return value 0-15 or -1 if fail
{
    if('0'<=HASCII && '9'>=HASCII) return(HASCII - '0');
    if('A'<=HASCII && 'F'>=HASCII) return(HASCII - 'A' + 10);
    return -1;
}


int hexcmd2bin(char *hexcmd, uint8_t *bincmd)   //processes a text buffer containing a MoT-format command 'string'  into a binary command buffer;
//  returns #of bytes in bincmd or -1 if format violation
{
    int i;
    uint8_t lonybble,hinybble,byte;
    uint8_t chk=0;
    if(hexcmd[0] != ':' ) return -1;    //bad format -- no start symbol
    for(i=1;i<MAXHEXBUF;) {
        if( (hexcmd[i]=='\n')| (hexcmd[i]=='\r')) break;//we're at the end of a command
        //here if parsing should continue
        if( ((hinybble=hex2bin(hexcmd[i]))>=0) && ((lonybble=hex2bin(hexcmd[i+1]))>=0) ) byte=16*hinybble+lonybble; //successful hex byte to bin conversion
        else return -2;                 //bad hex byte
        //here when the two characters of a hex byte are successfully converted into a value in 'byte'
        *bincmd++ = byte;               //add byte to the binary cmd string
        chk += byte;                    // and to the checksum (overflows are expected and desired)
        i += 2;                         //  then advance to convert the next byte
    }
    //here iff the text string lies between ':' and '\n' or '\r' and consists exclusively of hex bytes
    if ( chk != 0 ) return -3;          //bad checksum
    return i/2-1;                       //number of bytes in bindcmd, not including checksum
}

int ready_toggle(){
    if(inp_1 == 1 && inp_0 >= 3){
        inp_1 = 0;
        inp_0 = 0;
        return 1;
    }else{
```

```
  uint8_t *pbinbuf=USART3_binbuf;
          ^~~~~~~
Finished building: ../CM7/src/CM7_final_project.c

Building target: testy_CM7.elf
```

Figure 1. Workbench project window

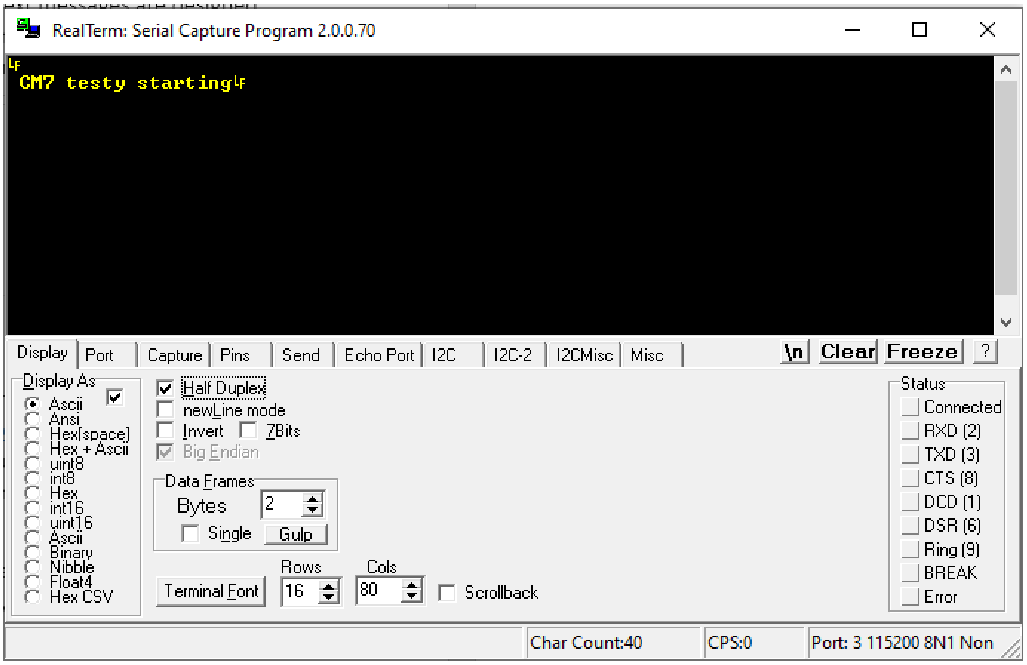
4. Click “Project” tab, click “Clean”, click “OK”
5. Click “Project” tab, click “Build all”. Look at the building console, if there are error messages and it says “Build finished”, the binary files are ready to be uploaded onto the board
6. Connect the board to PC through USB, making sure the power is on
7. Right click on the project folder in the Project Explorer, move mouse to “Target” and select “Program chip”.
8. Select both binaries in the “binaries” window, click “OK”. Check the building Console, Testy is ready to be used if it says “Build finished”.



```
17:46:45 Build Fin
```

Figure 2. Program Board

9. Please read to the "User Interface" section of "Testy Maintainer" document for steps how to setup Realterm for serial communication between PC and the board, as well as what the user will see once valid and invalid commands are sent.
10. Before pressing the reset button to restart Testy, make sure read the "Reminder/Limitation" section of "Testy Maintainer" document to make proper setup for some device functions
11. Press the black reset button on the Nucleo board, if user sees the following window on Realterm, great! It is ready to use! Try to enter some valid commands



RealTerm: Serial Capture Program 2.0.0.70

```
CM7 testy starting
```

Display As: ● Ascii ☑
☑ Half Duplex
□ newLine mode
□ Invert □ 7Bits
☑ Big Endian
Data Frames: Bytes 2; □ Single
Rows 16; Cols 80; □ Scrollback
Status: □ Connected □ RXD (2) □ TXD (3) □ CTS (8) □ DCD (1) □ DSR (6) □ Ring (9) □ BREAK □ Error

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

Figure 3. Testy is ready to use

Testterm

Testterm is executable application that will establish serial communication with board and execute prewritten scripts. For more details about how to write test scripts and how to run testterm, please read "_readme_testterm.pdf" in the test_script folder. Make sure do not write the reset command in the test script as it will crash the application. Instead, user should manually press the reset button for restarting Testy. Also, turn Realterm off when running testterm as one serial communication port can only connect to one software at a time.

Suggestions

To verify generated signals, if the user just wants to see the shape and period of the signal, using Arduino's plotter and monitor is very satisfied. For actual voltage reading, oscilloscope and multimeter are more accurate as there is a significant voltage reading offset between what is outputted from the board and what is read from the Arduino analog pins.