ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

ОДСЕК ЗА ЕЛЕКТРОНИКУ

ПРОЈЕКТНИ ЗАДАТАК

32-битни микроконтролери и примена

Професор:
др Драган Васиљевић

Студент:
Пантелија Брајић 12/3094
panta1brajic@hotmail.com

Београд, 2013.

Ова страница је намерно остављена празна

Садржај

Ова страница је намерно остављена празна

1. Увод

У извештају пројектног задатка представљен је поступак израде пороjекта из предмета 32-битни микроконтролери и примена, који се похађа на мастер студијама Одсека за електронику Електротехничког факултета у Београду. Пројекат подразумева употребу микроконтролера *STM*32*F*100*RB* произвођача *ST Semiconductors*, који се налази на развојном окружењу *STM*32*FVDISCOVERY* [1]. Језгро микроконтролера чини *ARM Cortex-M*3 процесор. Компоненте прикључене на развојно окружење су *TFT* диспјеј са екраном осетљивим на додир, контролер додирног панела и компонента са потенциометрима, и заједно чине целину. При развоју апликације коришћен је програмски пакет *IAR Embedded Workbench for ARM 6.40 Kickstart* који служи као комплетно развојно окружење за писање кода, симулацију и дебаговање апликације. Апликација је писана за радно окружење *embOS v*3.86 оперативног система који је намењен за рад у реалном времену. Инсталирана је пробна верзија овог оперативног система која је ограничена на највише три таска (процеса).

У првом поглављу дата је поставка пројектног задатка. У другој глави приказан је хардвер који се користи за имплементацију пројектог задатка као и концепт његове реализације. У трећој глави урађен је кратак осврт на пројекат. На крају је дат списак коришћене литературе.

1.1. Спецификација задатка

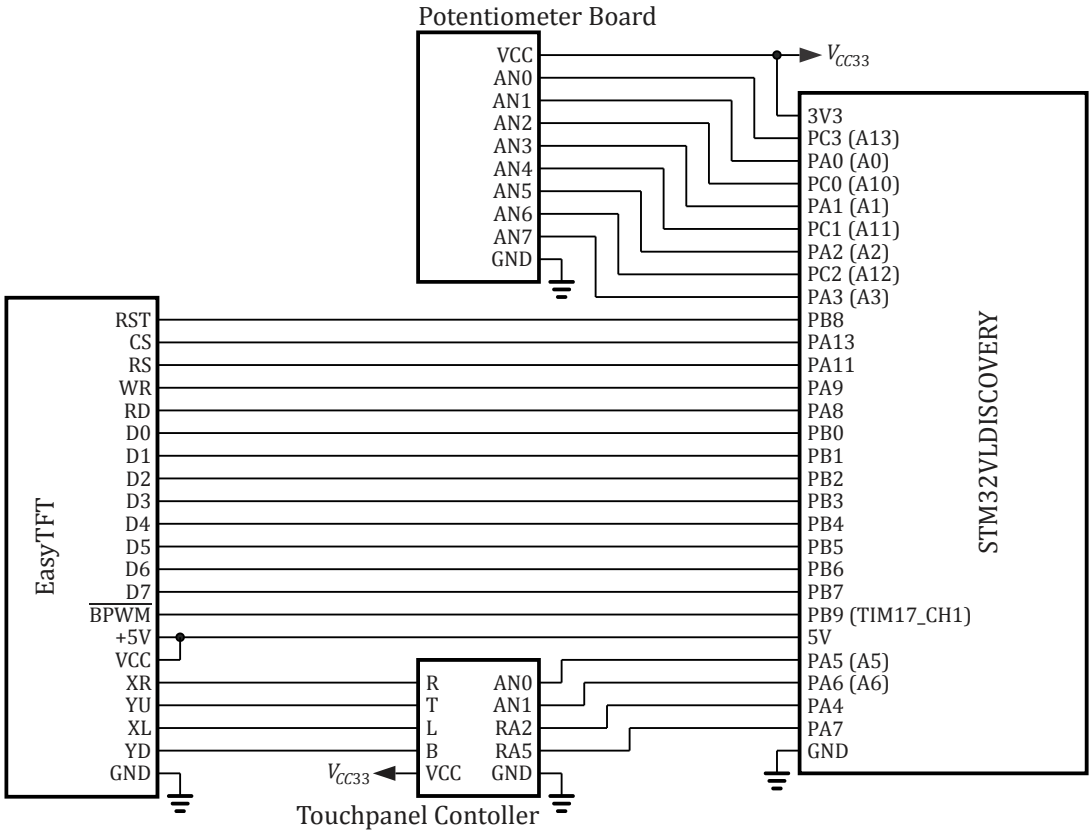
Потребно је написати програм којим се очитавају вредности са осам напонска канала и очитане вредности се исписују на *TFT* диспрел (графички) са одговарајућим пропратним текстом. Стартовање аквизације и контрола учестаности узорковања се врши тастерима преко додирног панела. Учестаност узорковања је од 0.5 Hz до 9.5 Hz.

Ова страница је намерно остављена празна

2. Реализација задатка

2.1. Опис система

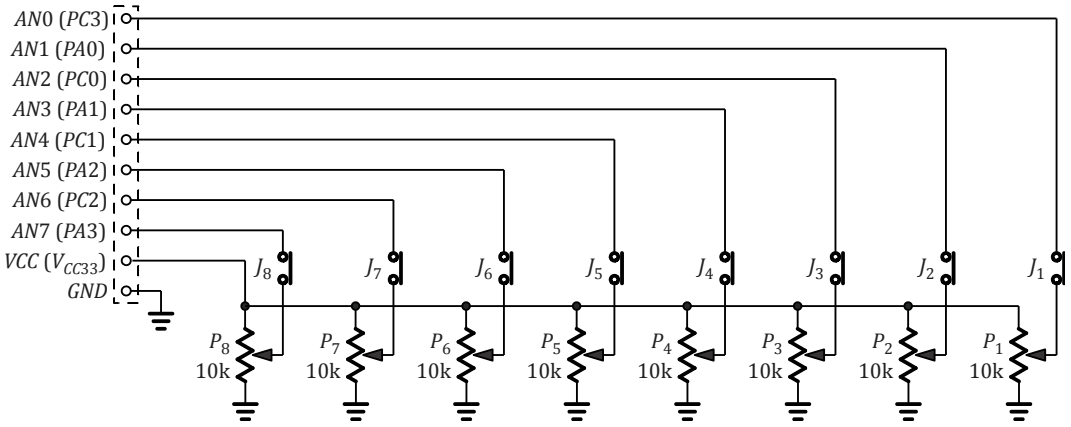
На слици 2.1 приказана је појендостављена блок шема развојног система *STM*32*FVDISCOVERY* са истакнутим везама између микроконтролера и периферија релевантних за овај пројектни задатак. На слици није приказан црни тастер који је намењен за ресетовање система, а налази се на развојном систему. Плави тастер, који такође није приказан, је тастер опште намене и везан је једним крајем за V_{CC33}, а другим крајем преко отпорника $R_{22} = 0\ \Omega$ за пин $PA0$. На пин $PA0$ доведен је и аналогни сигнал који може да буде везан на потенцијал масе. Због тога није дозвољена употреба плавог тастера док је систем под напајањем.



Слика 2.1. Развојни систем *STM*32*VLDISCOVERY* са *STM*32*F*100*RB* микроконтролером и релевантним периферијама

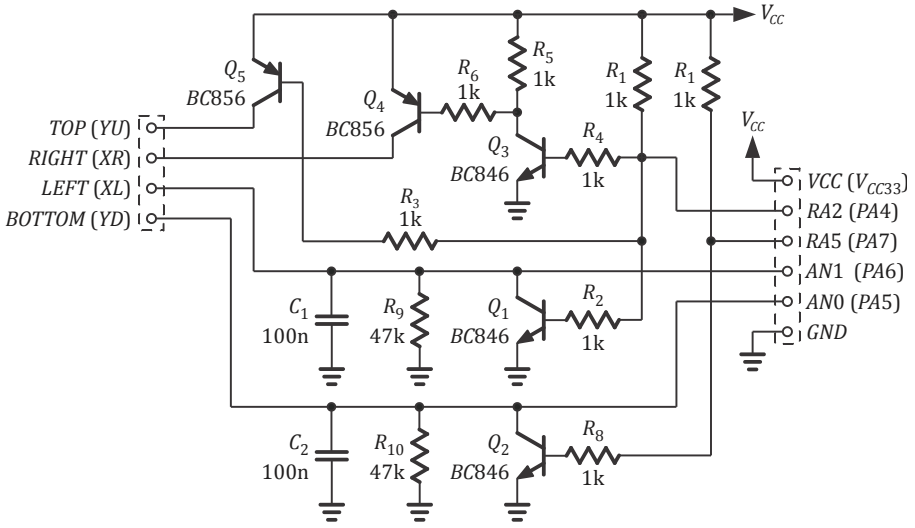
Следи кратак опис функционисања појединих блокова.

Компонента *Potentiometer Board* је намењена за генерисање осам аналогних сигнала чије се вредности крећу у границама напона напајања. Шематски приказ компоненте приказан је на слици 2.2. Потенциометрима од P_1 до P_8 подешава се напон одговарајућег канала. Краткоспојницима од J_1 до J_8 могуће је прекинути везу канала са потенциометром.



Слика 2.2. Шематски компоненте *Potentiometer Board*

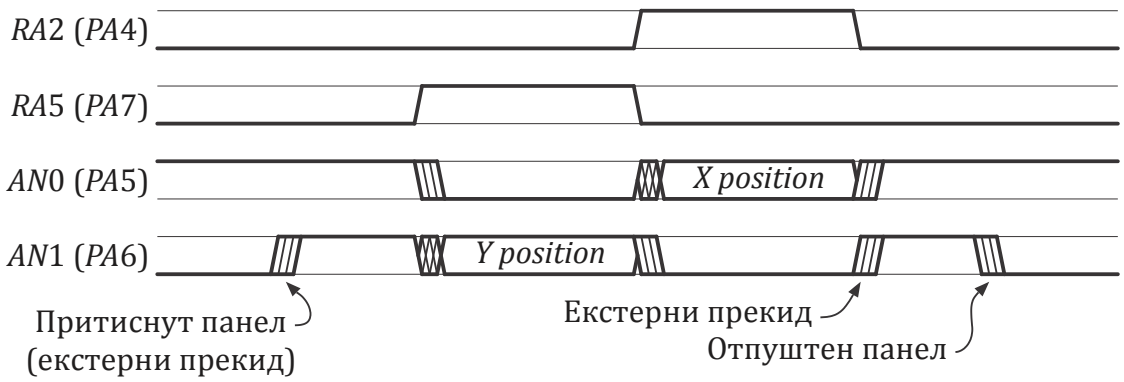
Компонента *Touchpanel Controller* је намењена за драјвовање прикључака додирног панела ради очитавања потиције додира. Шематски приказ компоненте приказан је на слици 2.3.



Слика 2.3. Шематски приказ драјвера за додирни панел

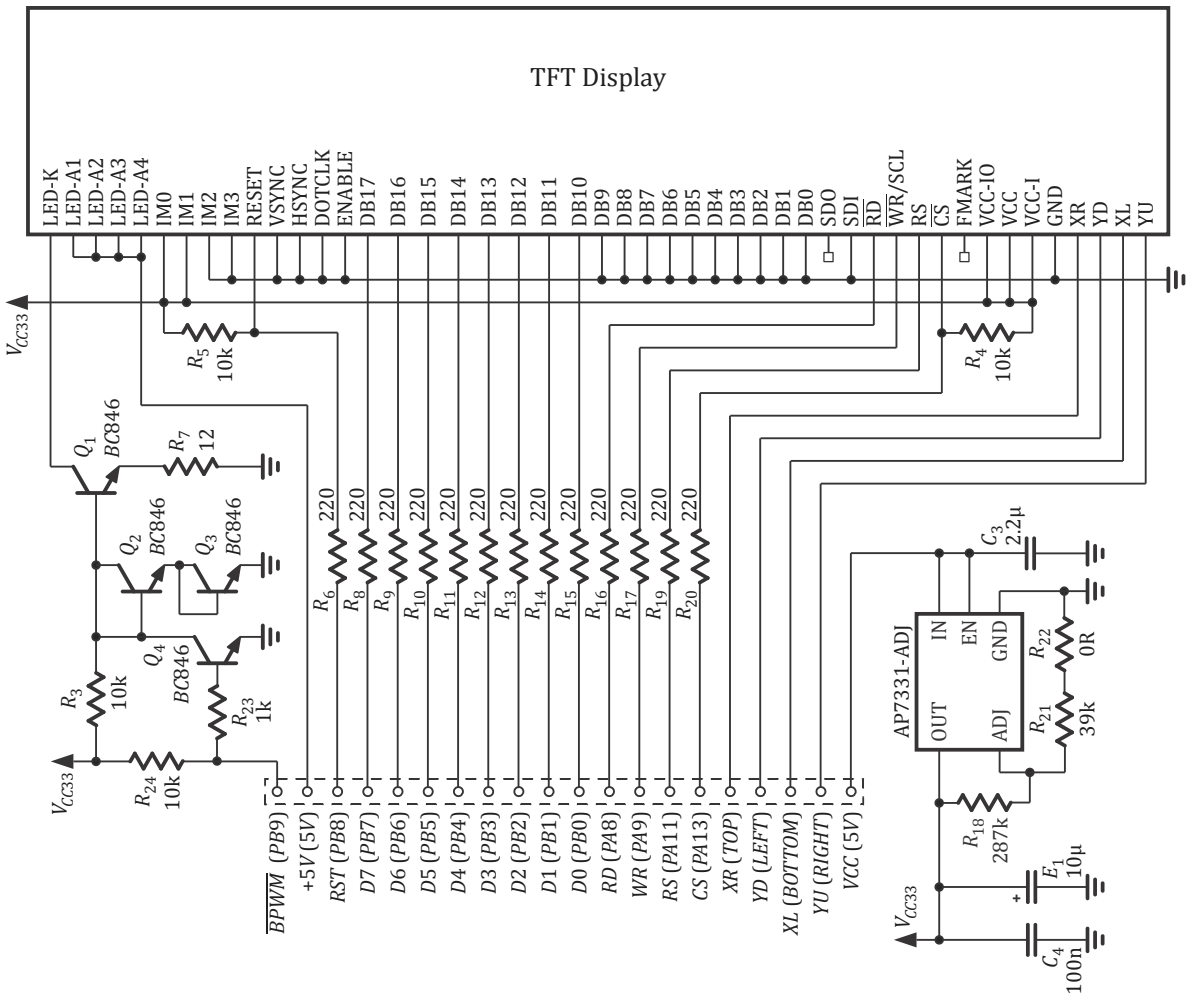
На слици 2.4 приказан је временски дијаграм сигнала за правилно очитавање позиције додира панела. На самом почетку, сигнли *RA*2 и *RA*5 имају вредност логичке нуле. Тада је напон прикључка *YU* додирног панела близу V_{CC33}. Отпорност фолије додирног панела за једну осу је реда величине неколико стотина ома, зато је и напон прикључка *YD* близу V_{CC33}. Прикључак *AN*1 је на потенцијалу масе због отпорника R_9. Када се екран притисне, фолије у *X* и *Y* правцу се споје, па се напон на прикључку *AN*1 промени на V_{CC33}. Ова узлазна ивица сигнала *AN*1 подешена је да буде спољашњи прекид и служи да обавести систем да је екран притиснут. Након тога се врше очитавање позиције додира. Потребно је

дебаусирати сигнал за детекцију притиска панела јер се фолије приликом спајања понашају као реали прекидач. Вредности појединих трајања на слици 2.5 одређена су временским константама прикључака.



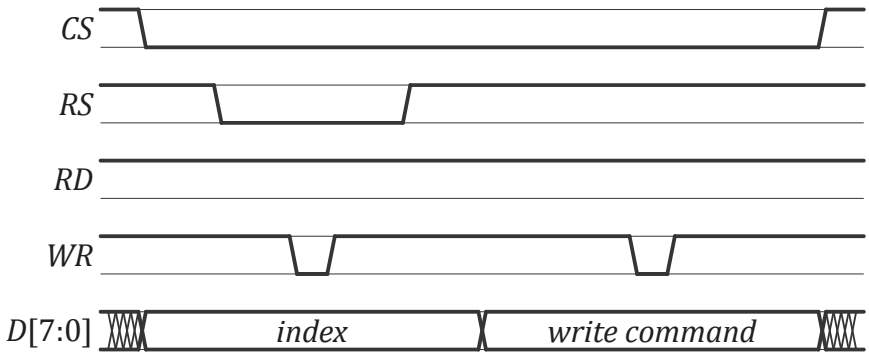
Слика 2.4. Временски дијаграм сигнала
приликом очитавања координате додира

На слици 2.5 је приказан интерфејс *TFT* дисплеја [8]. Подаци се преносе паралелно преко осам линија *D*[7:0] које су повезане са највиших осам линија магистрале података *DB*[17:10] контролера дисплеја *HX*8347-*D* [7] (паралелни 8-битни интерфејс, тип 2).

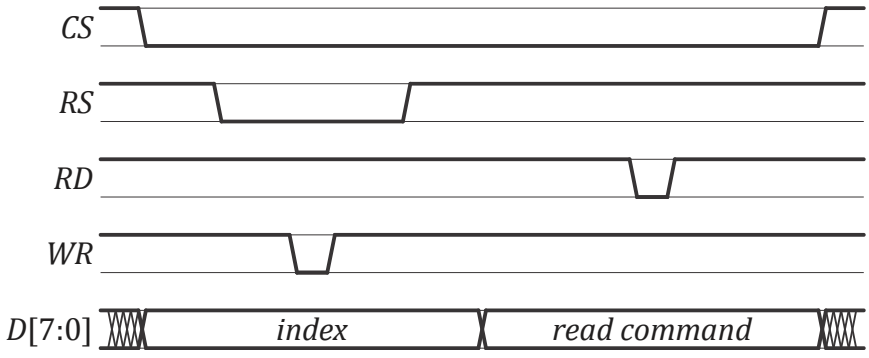


Слика 2.5. Интерфејс *TFT* дисплеја

Помоћу прикључка $\overline{BPWM}$ подешава се интензитет позадинског светла. На $\overline{BPWM}$ прикључак доводи се *PWM* сигнал са тајмера *TIM*17_*CH*1. Пин *PB9* је конфигурисан да ради као излаз са отвореним дрејном. Прикључак *RST* представља сигнал за ресет дисплеја. Сигнали *RS*, *CS*, *WR*, *RD* служе за упис команде или параметра у контролер дисплеја. Временски дијаграм сигнала за упис у и читање из *TFT* дисплеја приказан је на слици 2.6. Вредности појединих трајања на слици 2.6 могу се пронаћи у каталогу дисплеја.



(а)



(б)

Слика 2.6. (а) Упис команде у регистар и
(б) читање податка из регистра

Критични временски параметри за функцију уписа обично су краћи од периоде системског такта па се функција уписа може извршити у неколико периода сигнала такта. Функција читања траје знатно дуже од функције уписа и морају се убацивати кашњења.

2.2. Иницијализација система

По укључењу напајања односно екстерном ресету система, неопходно је иницијализовати портове микроконтролера и коришћених периферијских модула. На самом почетку врши се иницијализација оперативног система позивом функције *OS_InitKern*(). Позивном функције *OS_InitHW*() иницијализује се хардвер неопходан за оперативни систем и искључује се *watchdog* тајмер да не би прекинуо процес иницијализације. *Watchdog* тајмер остаје искључен и након иницијализације пошто се не користи у овом пројекту. Након тога врши се

иницијализација контроле системског такта, тј. довођење сигнала такта периферијама које се користе у овом пројекту, након чега се иницијализују периферије. На крају се иницијализује *TFT* дисплеј. Затим се креирају објекти кернела – два таска и један бројачки смафор. По завршетку иницијализације целог оперативног система, контола програма препушта се кернелу и почиње мултитаскинг позивом функције *OS_Start*().

Иницијализација контроле системског такта

Пре иницијализације било које периферије, потребно је довести такт сигнал периферији. Периферије које се корсте у пројекту су: *GPIO* портови *A*, *B* и *C*, аналогно-дигитални конвертор, тајмери *TIM*1 и *TIM*17, *DMA* контролер и алтернатвне функције. Све периферије су повезане на *APB2* магистралу осим *DMA* контролера који је повезан на *AHB* магистралу.

Иницијализација портова

По ресету, сви портови микроконтролера су дефинисани као улазни, па је потребно експлицитно дефинисати смер само оних портова (пинова) који се користе као излазни.

Пинови 0, 1, 2, 3, 5 и 6 порта *A* се користе за очитавање аналогних сигнала са потенциометара и додирног панела и њихов смер је аналогни улазни. Понови 4, 7, 8, 9, 11 и 13 порта *A* користе се за управљање контролера додирног панела и *TFT* дисплеја, па је њихов смер излази. Главна функција, након екстерног ресета, додељена пину 13 порта *A* је серијски порт за дебаговање (*SWJ-DP*). Неопходно је ремапирати пин на његову алтернативну функцију, тј. да је пин у служи излазног порта. Међутим, у том случају је у потпуности онеспособљен порт за дебаговање те није могуће узети било какав податак из интерних регистара.

Пинови 0, 1, 2, 3, 5, 6, 7 и 8 порта *B* су конфигурисани као излазни пошто се користе за контролу *TFT* дисплеја. Читање података из дисплеја се не примењује у пројекту. Главна функција, након екстерног ресета, додељена пину 4 порта *B* је серијски порт за дебаговање (*SWJ-DP*). Неопходно је ремапирати пин на његову алтернативну функцију, тј. да служи као излазни порт. Пин 9 је такође потребно ремапирати на његову алтернативну функцију, тј. да служи као излаз тајмера *TIM*17_*CH*1. Тајмер *TIM*17 се користи за генерисање *PWM* сигнала који регулише позадинско осветљење дисплеја.

Пинови 0, 1, 2 и 3 порта *C* се користе за очитавање аналогних сигнала са потенциометара и њихов смер је аналогни улазни.

Иницијализација аналогно-дигиталног конвертора

Аналогно дигитални конвертор врши аквизицију сигнала са осам потенциометара и аквизицију још два сигнала са додирног панела. Аналогни сигнали са потенциометара поређани су у регуларном секвенцеру и њихово скенирање отпочиње на узлазну ивицу *PWM* сигнала *TIM*1_*CC*1 тајмера *TIM*1. Омогућен је *DMA* пренос за регуларне конверзије. Аналогни сигнали са додирног поређани су у инјектованом секвенцеру и њихова конверзија отпочиње са софтверски генерисаним сигналом. Након иницијализације, врши се калибрација аналогно-дигиталне конверзије.

Иницијализација тајмера

Тајмер *TIM*1 користи се за стартовање конверзије секвенце регуларних канала *AD* конвертора. Тајмер је конфигурисан да ради у *PWM* режиму, те узлазна ивица сигнала са компаратора *CC*1 отпочиње конверзију. Фреквенција *PWM* се мења тако што се мења вредност периоде у *ARR* регистру. На почетку је онемогућен бројач, тако да је сигнал на излазу тајмера увек логичка нула. Тајмер се стартује притиском на одговарајуће поље на додирном панелу.

Тајмер *TIM*17 користи се за регулисање позадинског осветљења дисплеја. Тајмер је конфигурисан у *PWM* режиму. Фактор испуњености се мења тако што се мења вредност у *CCR* регистру. Периода сигнала је постављена на 100 Hz да би се избегао ефекат треперења слике.

Иницијализација контролера прекида

Постоје два извора прекида. Један од њих је екстерни прекид *EXTI9_5* додељен пину *PA*6, који се генерише притиском на додирни панел. У прекидној рутини екстерног прекида врши се инкрементирање бројачког семафора. Други прекид је прекид *DMA* контролера *DMA_TC*, који се генерише након завршетка пребацивања осам података из аналогно-дигиталног конвертора у меморију. Прекидни захтев *DMA* контролера постављен је да буде већег приоритета од екстерног прекида. Прекидна рутина *DMA* прекида сигнализира да су скениране нове вредности аналогних сигнала.

Оба таска морају да буду приоритета од 128 до 255 јер се користе функције оперативног система у прекидним рутинама. Уместо онемогућавања прекида када *embOS* ради атомске операције, ниво прекида процесора постављен је на 128. Уколико су прекиди приоритета мањег од 128, они ће оперативни систем и програм ће се заглавити у прекидној рутини на позиву неке од функција оперативног система.

Иницијализација екстерних прекида

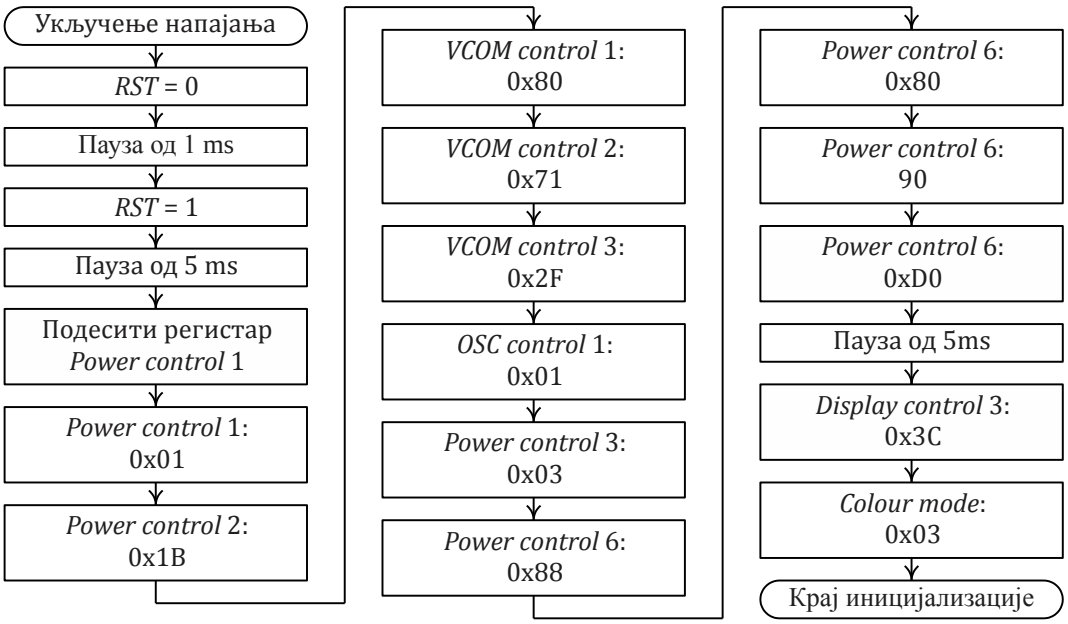
Екстерни прекид генерише се на узлазну ивицу сигнала са контролера додирног панела *AN*0 који је доведен на *PA*6. Екстерни прекид се генерише по други пут у случају када се завршава скенирање позиције додира и при том је додир и даље присутан (слика 2.4).

Иницијализација *DMA* контролера

DMA контролер служи за пребацивање податка из аналогно-дигиталног конвертора у меморију без интервенције процесора. На канал 1 *DMA* контролера доводи се захтев аналогно-дигиталног конвертора за *DMA* циклус. Неопходно је да се пребаци осам података из аналогно-дигиталног конвертора у меморију. Ти подаци представљају регуларне узорке сигнала са осам потенциометара. Након што се изврши пребацивање свих осам података, *DMA* контролер генерише захтев за прекид *DMA_TC*.

Иницијализација *TFT* дисплеја

Дијаграм тока иницијализације *TFT* дисплеја приказан је на слици 2.7. Затим се исписује се бела позадина и пропратни текст, што није наведено у дијаграму тока иницијализације.



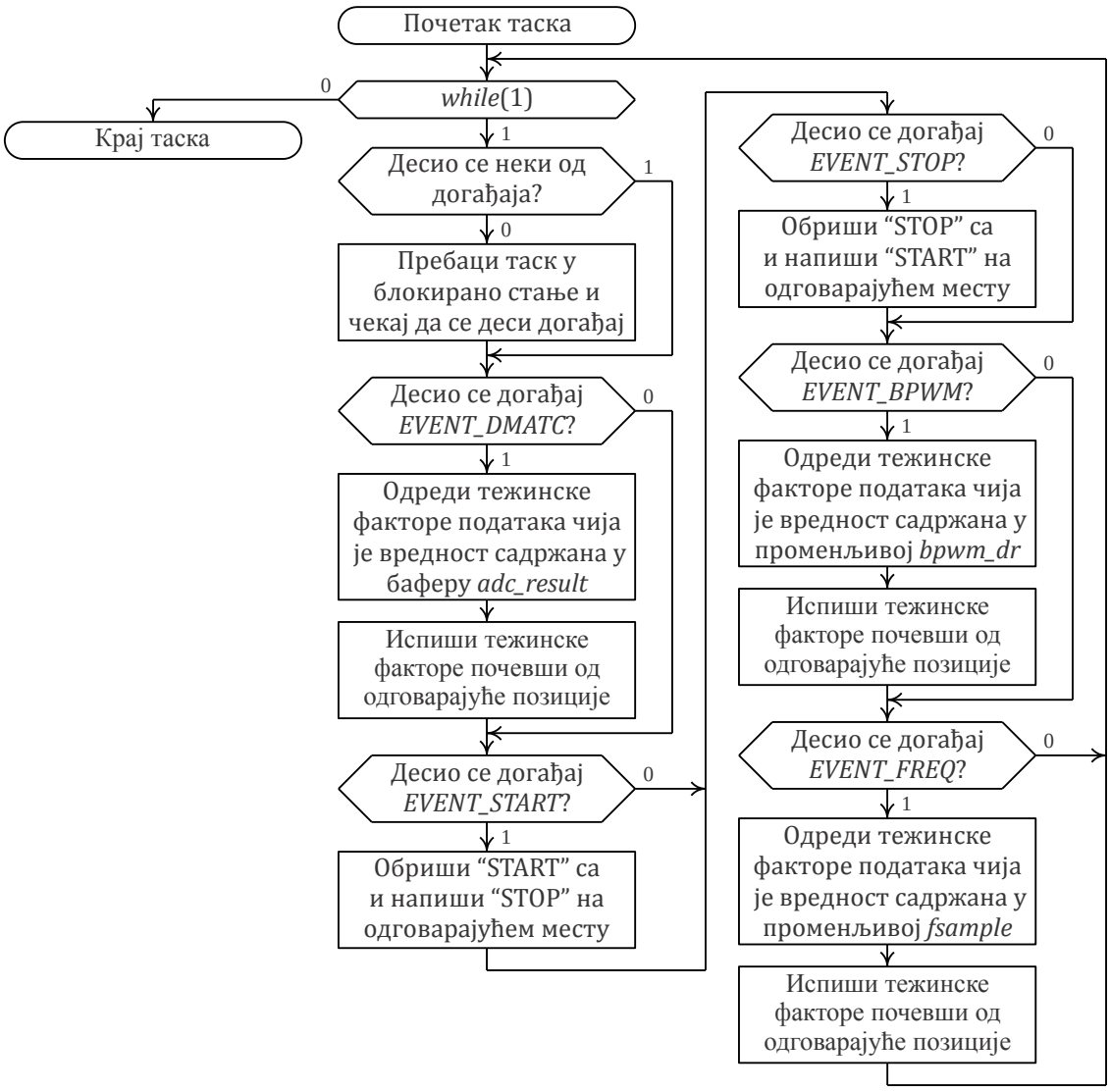
Слика 2.7. Процедура иницијализације дисплеја

2.3. Објекти кернела

У систему постоје два независна таска, *TFT_TASK* и *PEN_TASK*. Једносмерна комуникација између таскова постигнута је са четири догађа *EVENT_FREQ, EVENT_BPWM, EVENT_START* и *EVENT_STOP* које таск *PEN_TASK* сигнализира таску *TFT_TASK*. Прекидна рутина *DMA* контролера *DMA_Channel*1 такође сигнализира jedan догађај *EVENT_DMATC* таску *TFT_TASK*. Помоћу бројачког семафора *PEN_CSEMA* успостављена је синхронизација таста *PEN_TASK* и прекидне рутине *EXTI*9_5 екстерног прекида.

Таск за испис података на *TFT* дисплеј *TFT_TASK*

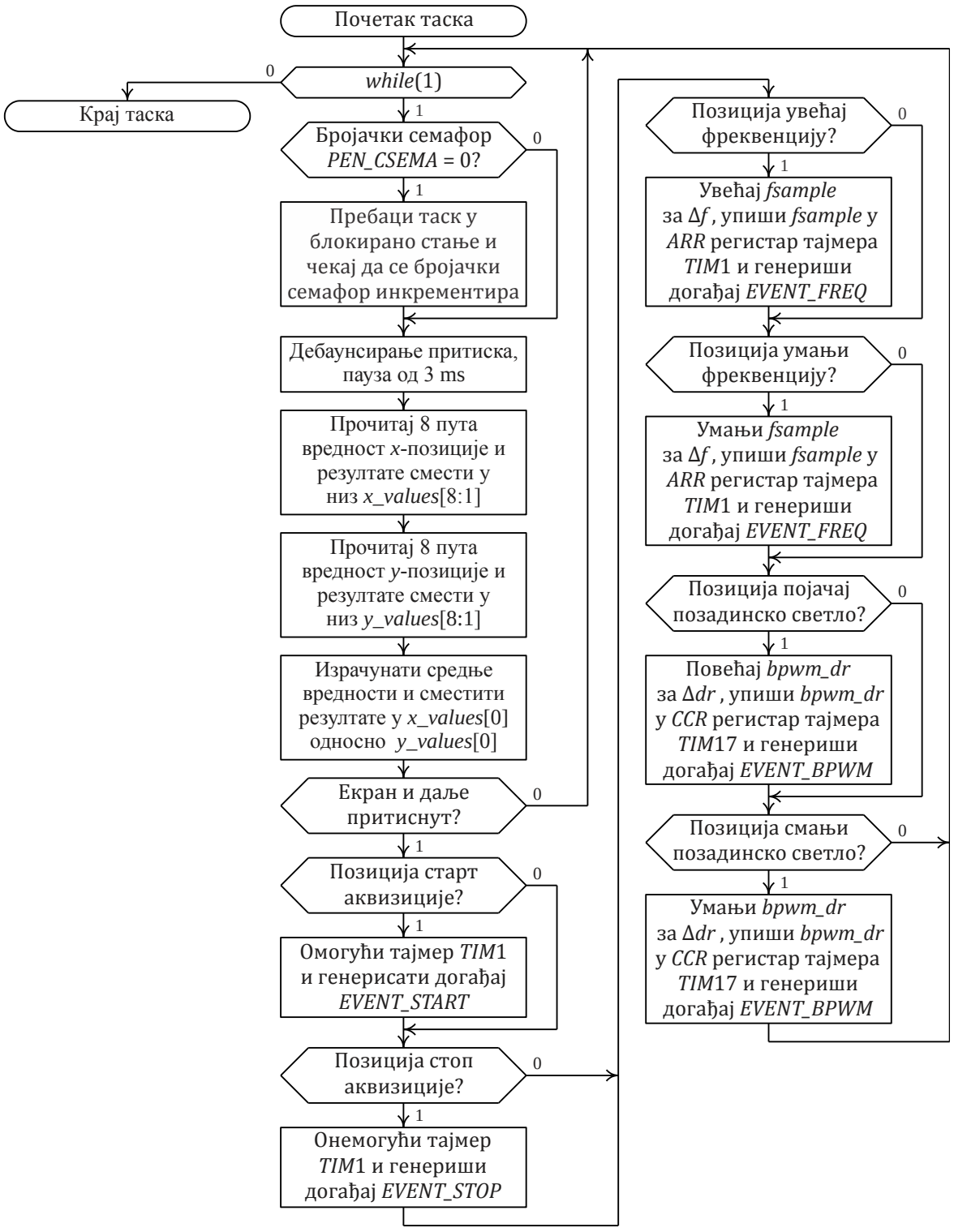
Таск *TFT_TASK* има улогу да исписује податке на дисплеј. Он чека у блокираном стању док му други таск или прекидна рутина не сигнализирају да су подаци које треба да испише на дисплеј освежени. Прекидна рутина *DMA* прекида сигнализира *EVENT_DMATC* ако су скениране нове вредности аналогних сигнала. Таск *PEN_TASK* сигнализира следеће: *EVENT_FREQ* ако је дошло до промене фреквенције семпловања; *EVENT_BPWM* ако је дошло до промене вредности интензитета осветљења дисплеја; *EVENT_START* ако је покренута аквизиција; *EVENT_STOP* ако је аквизивија заустављена. Релативна грешка која се уноси у резултат приликом одређивања тежинских фактора је у границама ±1%. Дијаграм тока извршавања таска приказан је на слици 2.8.



Слика 2.8. Дијаграм тока извршавања таска *TFT_TASK*

Таск за полирање додирног панела *PEN_TASK*

Таск *PEN_TASK* има улогу да одреди позицију контакта додирног панела. Таск чека у блокираном стању док му прекидна рутина *EXTI*9_5 не сигнализирају да је додирни панел притиснут тако што инкрементира бројачки семафор. У зависности од позиције додира, таск такође изврши сва неопходна израчунавања, остављајући таску *TFT_TASK* само да испише крајње резултате на дисплеј. Кашњење од 200 ms на крају таска представља некакав вид дебаунсирања и одређује полирање додирног панеса са фреквенцијом од око 5 Hz. Дијаграм тока извршавања таска *TFT_TASK* приказан је на слици 2.9.



Слика 2.9. Дијаграм тока извршавања таска *PEN_TASK*

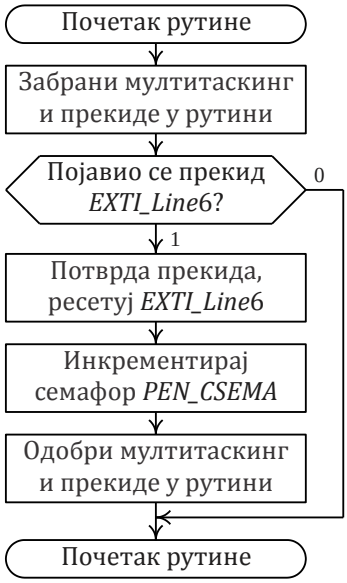
Прекидна рутина *DMA* контролера *DMA_Channel*1*_IRQHandler*

Дијаграм тока извршавања прекидне рутине *DMA_Channel*1*_IRQHandler* која опслужује прекид *DMA* контролера приказана је на слици 2.10.

Слика 2.10. Прекидна рутина *DMA_Channel*1_*IRQHandler*

Прекидна рутина *EXTI*9_5_*IRQHandler*

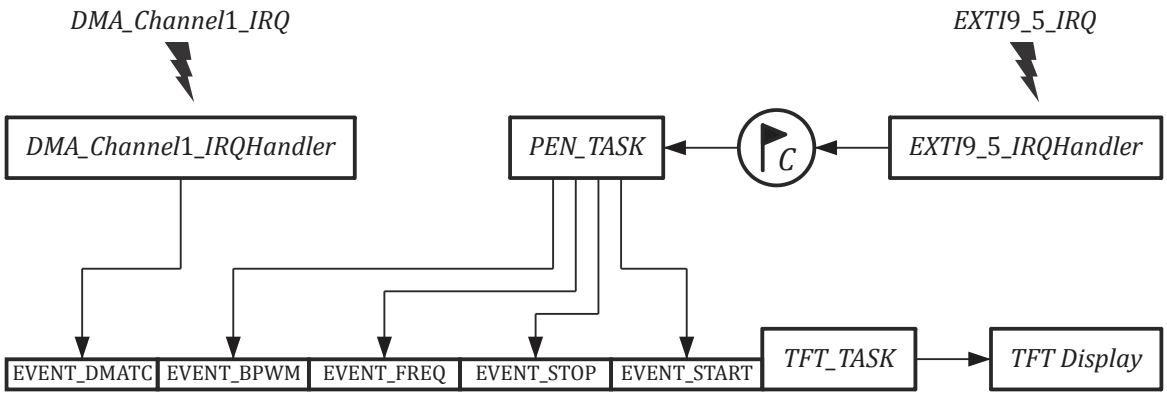
Дијаграм тока извршавања прекидне рутине *EXTI*9_5_*IRQHandler* која опслужује екстерни прекид приказана је на слици 2.11.



Слика 2.11. Прекидна рутина *EXTI*9_5_*IRQHandler*

2.4. Синхронизација и комуникација објеката кернела

Механизан комуникације и синхронизације активносит таскова приказан је на слици 2.12.



Слика 2.12. Дијаграм тока извршавања таска *PEN_TASK*

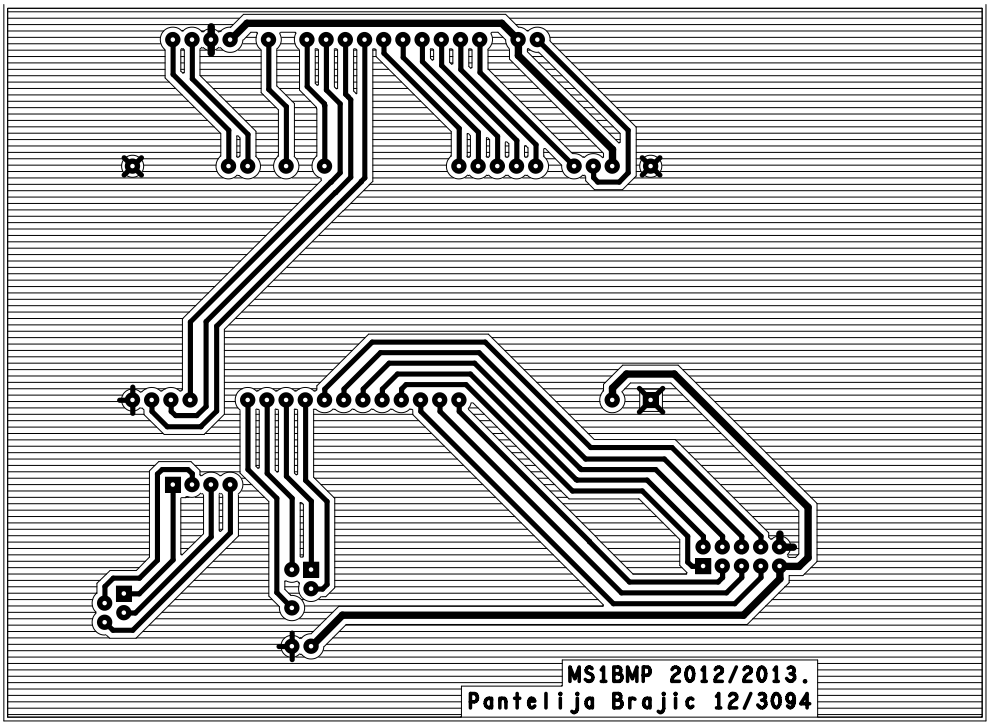
Таск *PEN_TASK* је таск највећег приоритета креће са извршавањем одмах након стартовања оперативног система. Блокира се на бројачном семафору јер је иницијална вредност бројачног семафора постављена на нулу. Након што се блокирао таск *PEN_TASK*, таск *TFT_TASK* креће са извршавањем. Таск *TFT_TASK* се блокира док чека на један од догађаја. Када се притисне додирни панел, генерише се екстерни прекид *EXTI*9_5. Прекидна рутина додељена екстерном прекиду инкрементира бројачки семафор. Тада таск *PEN_TASK* прелази у стање извршавања као таск са највећим приоритетом. У зависности од позиције додира, такск *PEN_TASK* генерише неки догађај или покреће тајмер *TIM*1.

Узлазна ивица *PWM* сигнала *TIM*1_*CC*1 тајмера *TIM*1 покреће скенирање осам регуларних канала. Након сваке конверзије генерише се *DMA* захтев. *DMA* контролер пребацује конвертовани податак из периферије у меморију. Након што пребаци свих осам података, *DMA* контролер шаље прекидни захтев *DMA_IT_TC*. Прекидна рутина *DMA* контролера генерише догађај *EVENT_DMATC* који сигнализира да су нови узорци спремни за приказивање.

Ова страница је намерно остављена празна

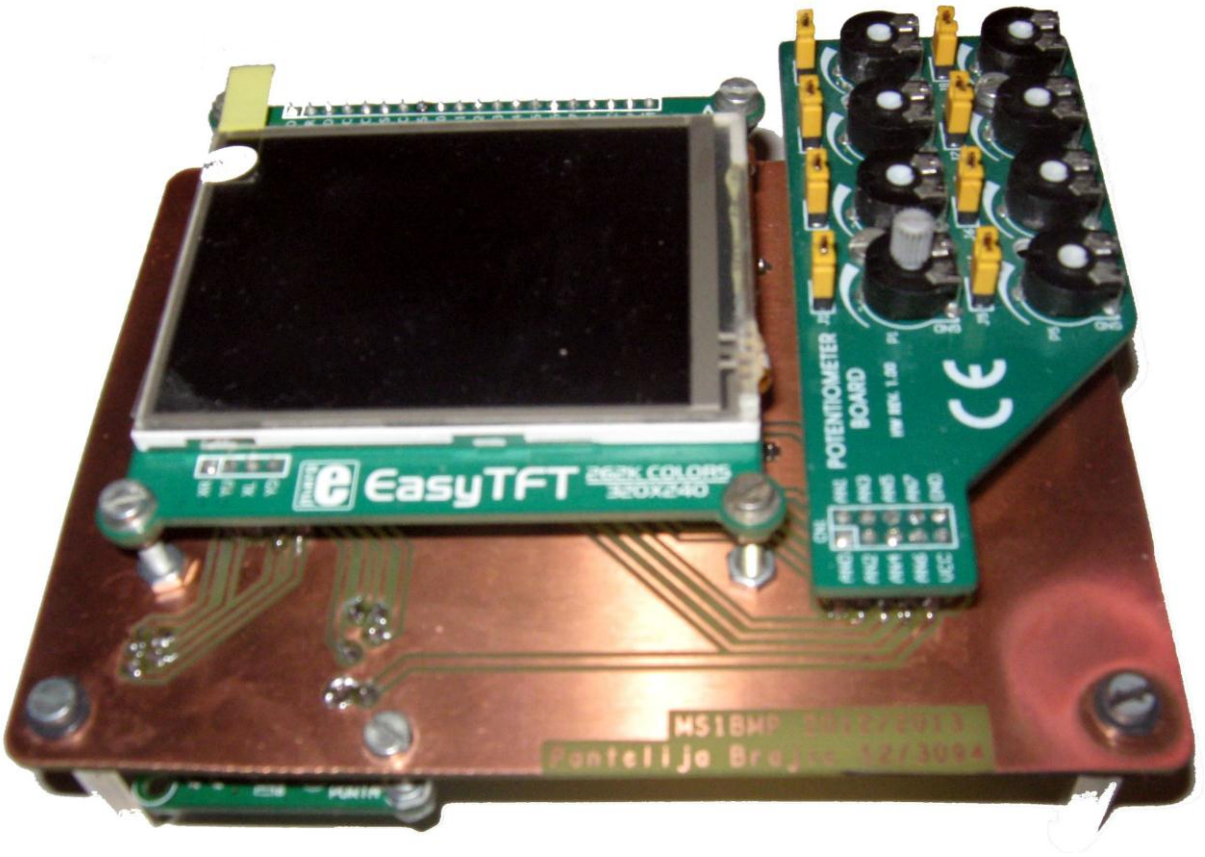
3. Фотографије система

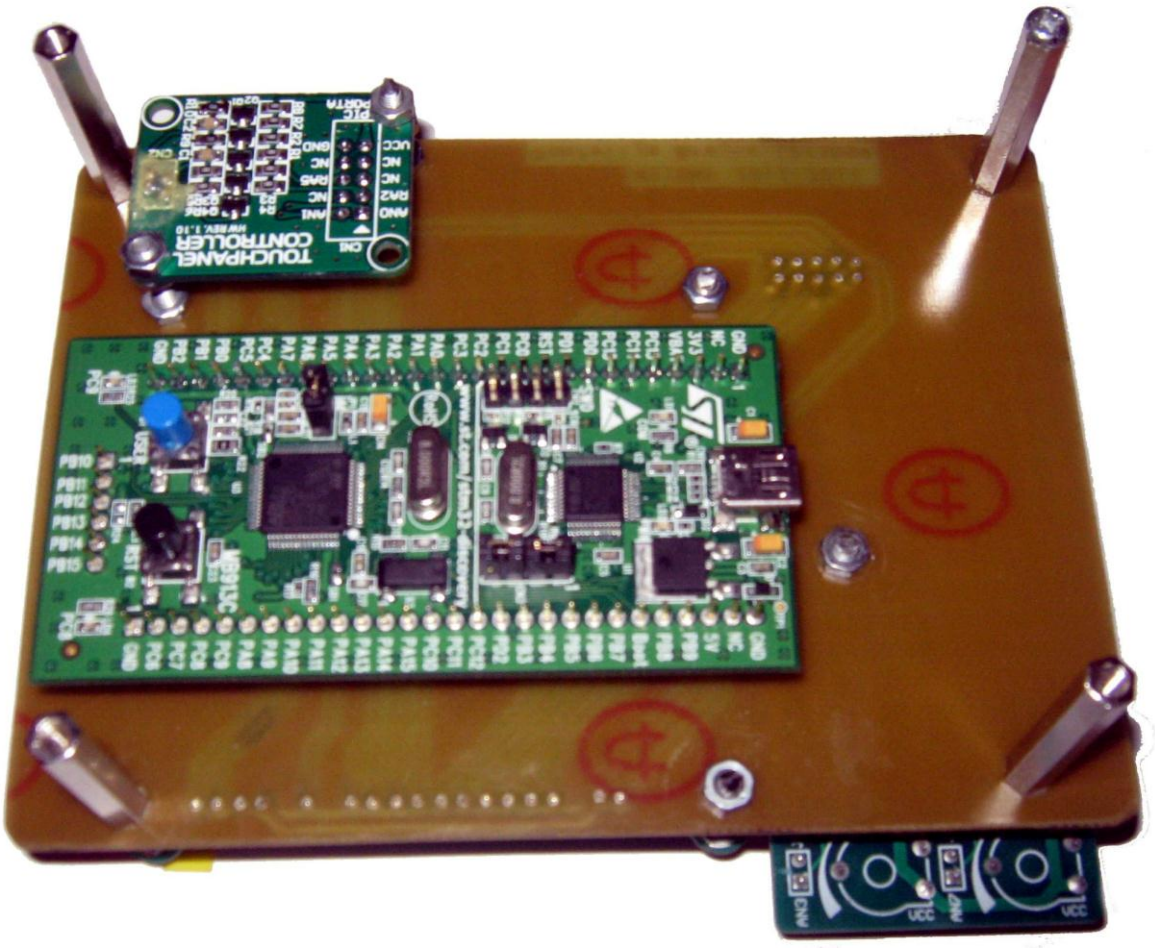
На слици 3.1 приказана је штампана плоча на коју су касније компоненте учвршћене и међусобно повезане.

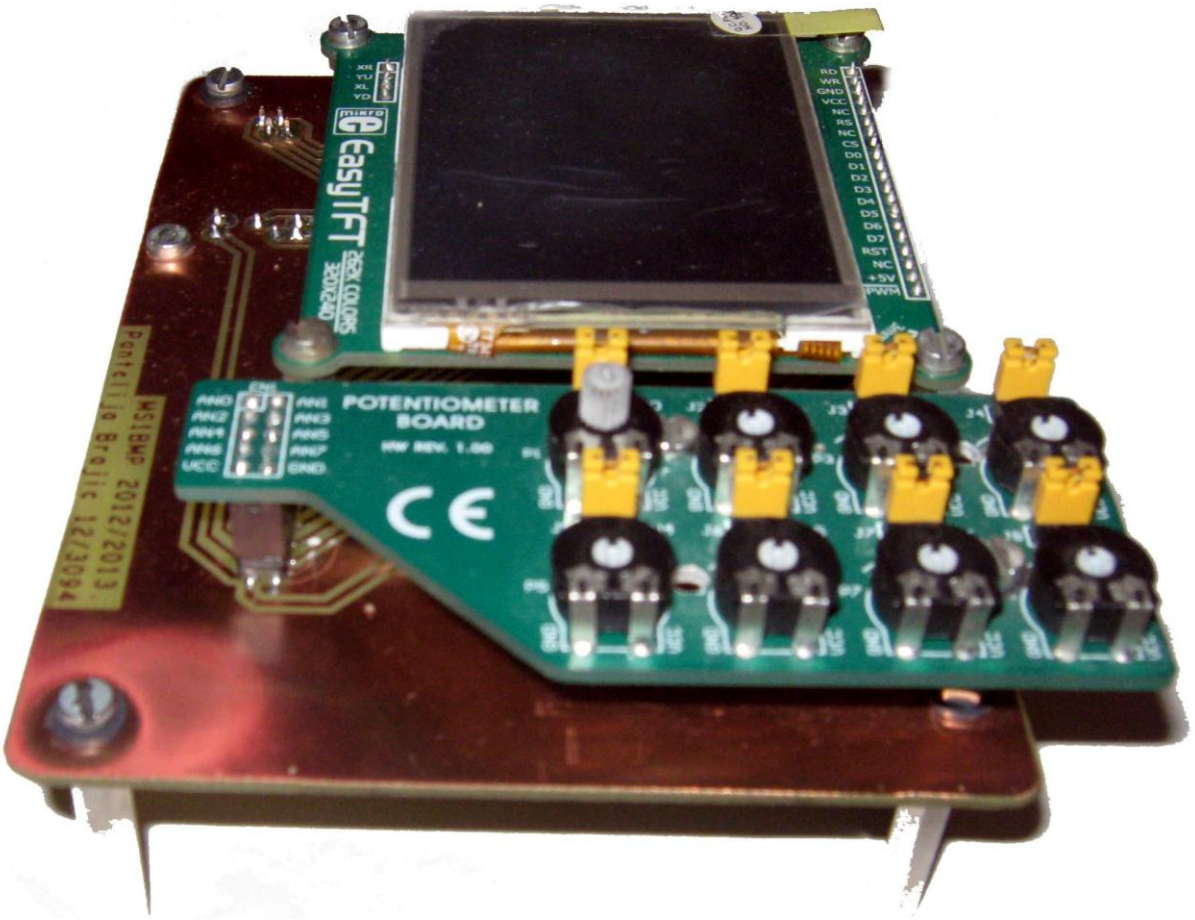


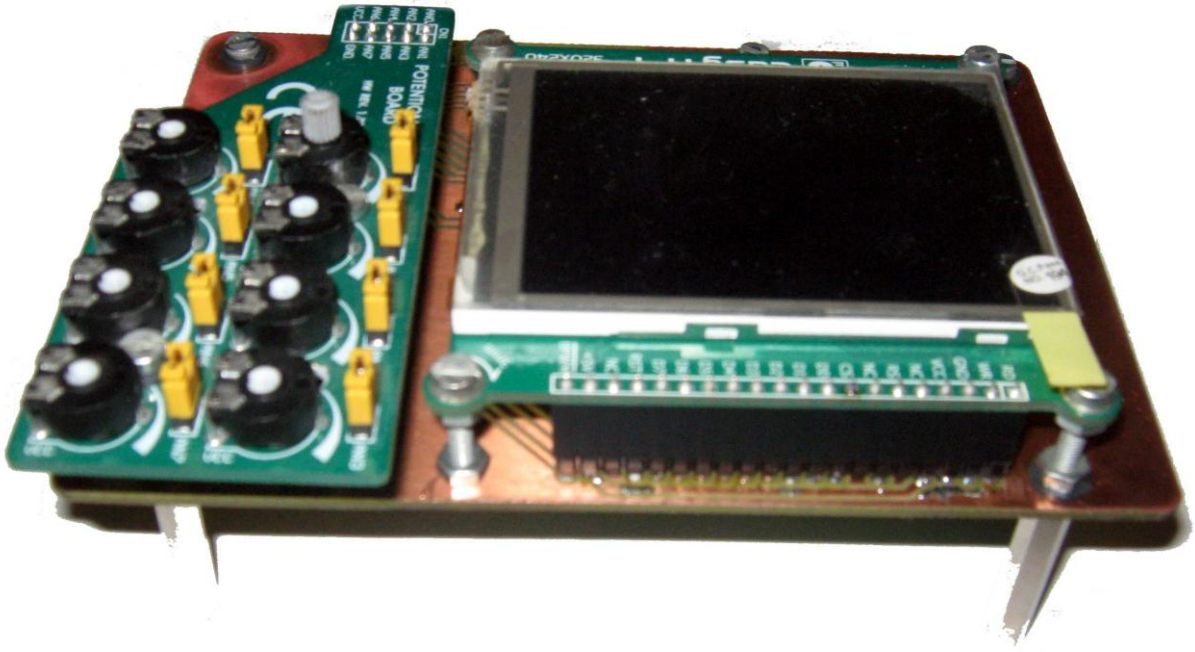
Слика 3.1. Штампана плоча

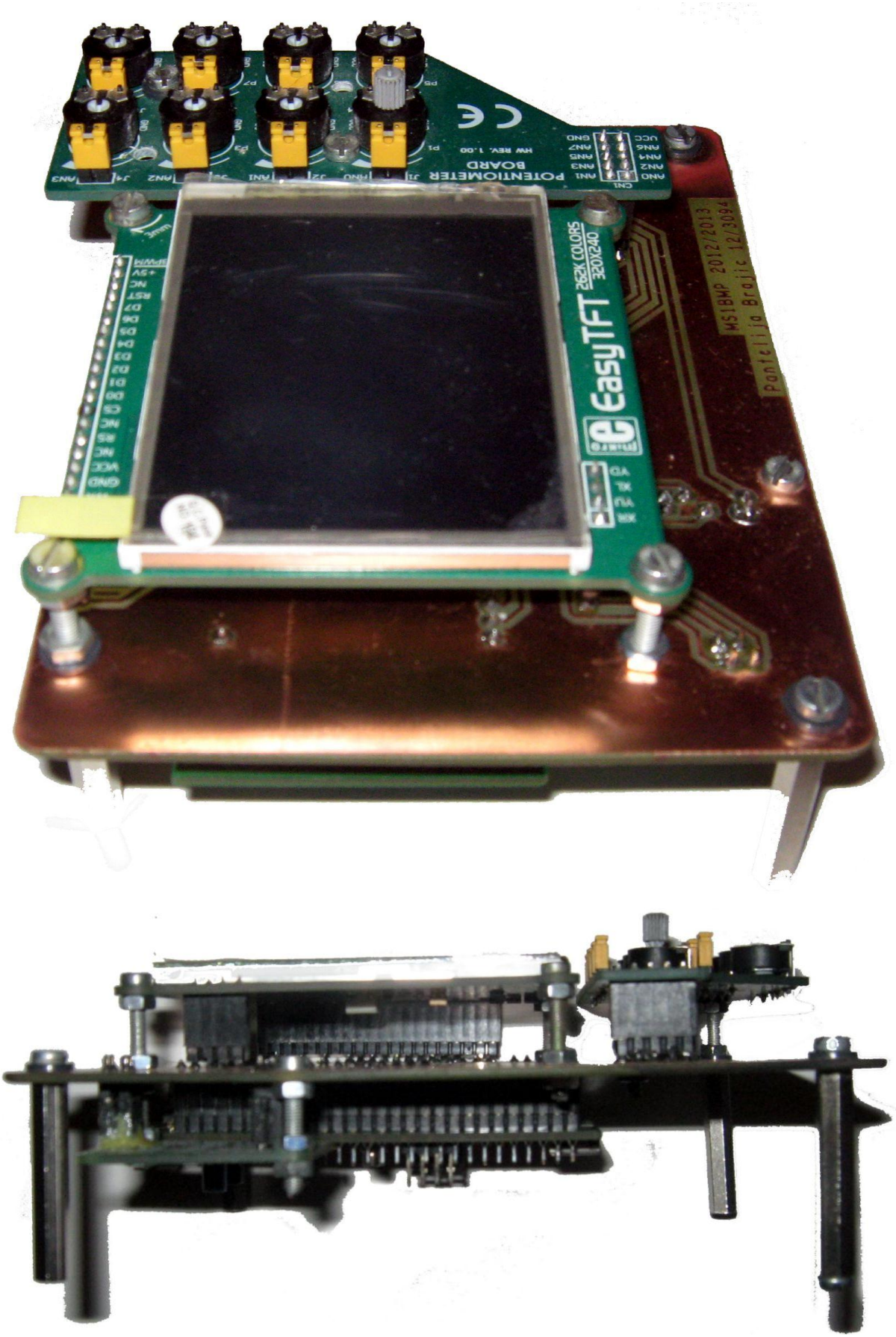
У наставку приказано је неколико фотографија готовог уређаја.

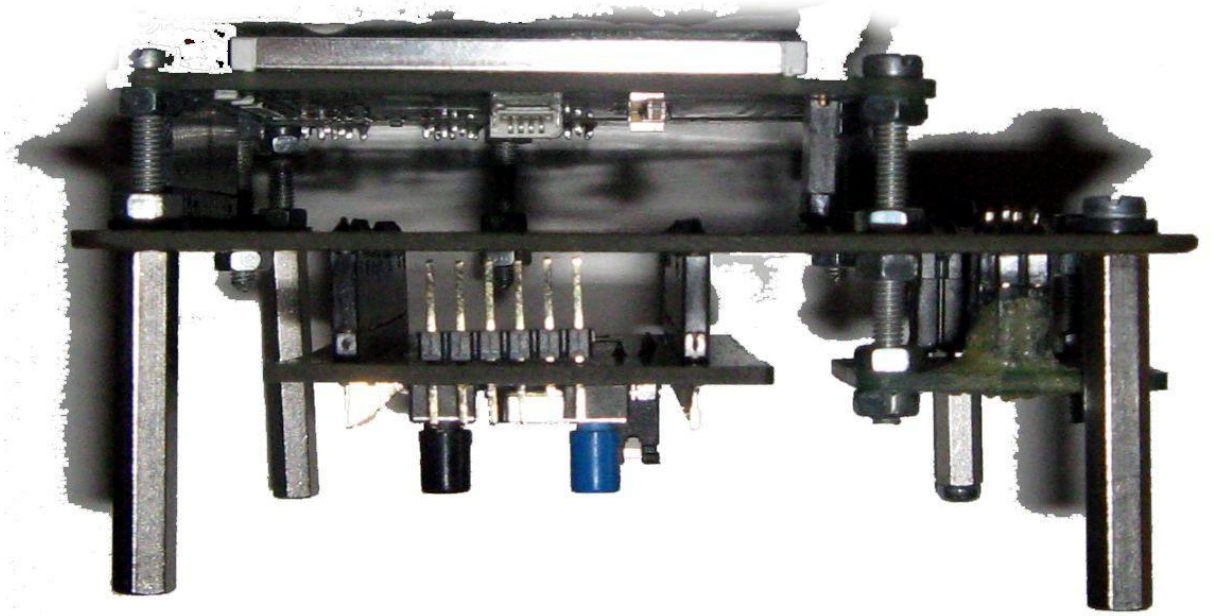

EasyTFT
POTENTIOMETER BOARD

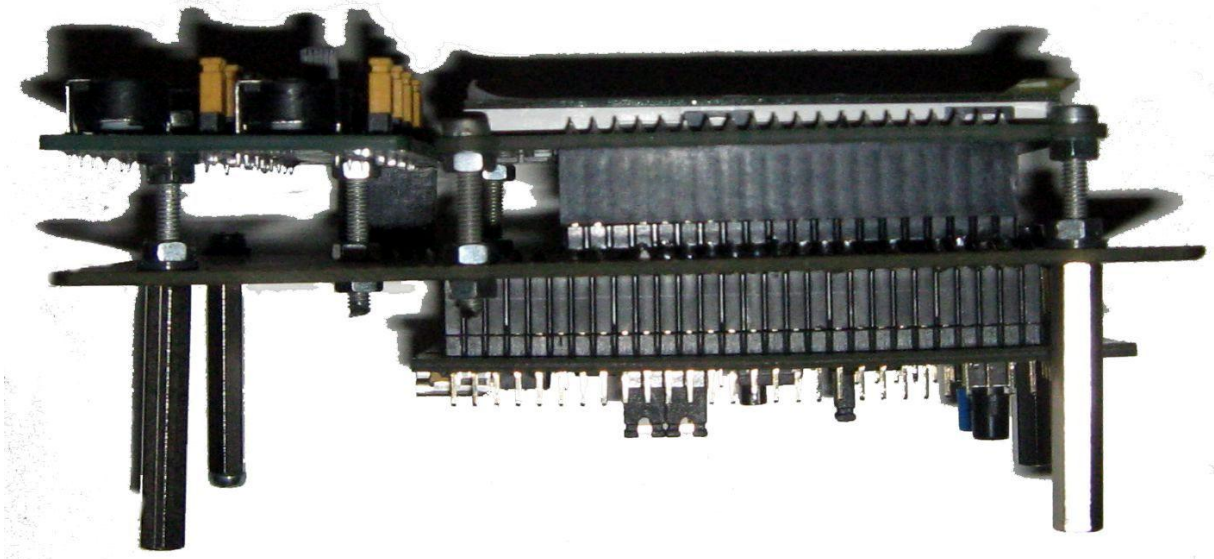

TOUCHPANEL CONTROLLER

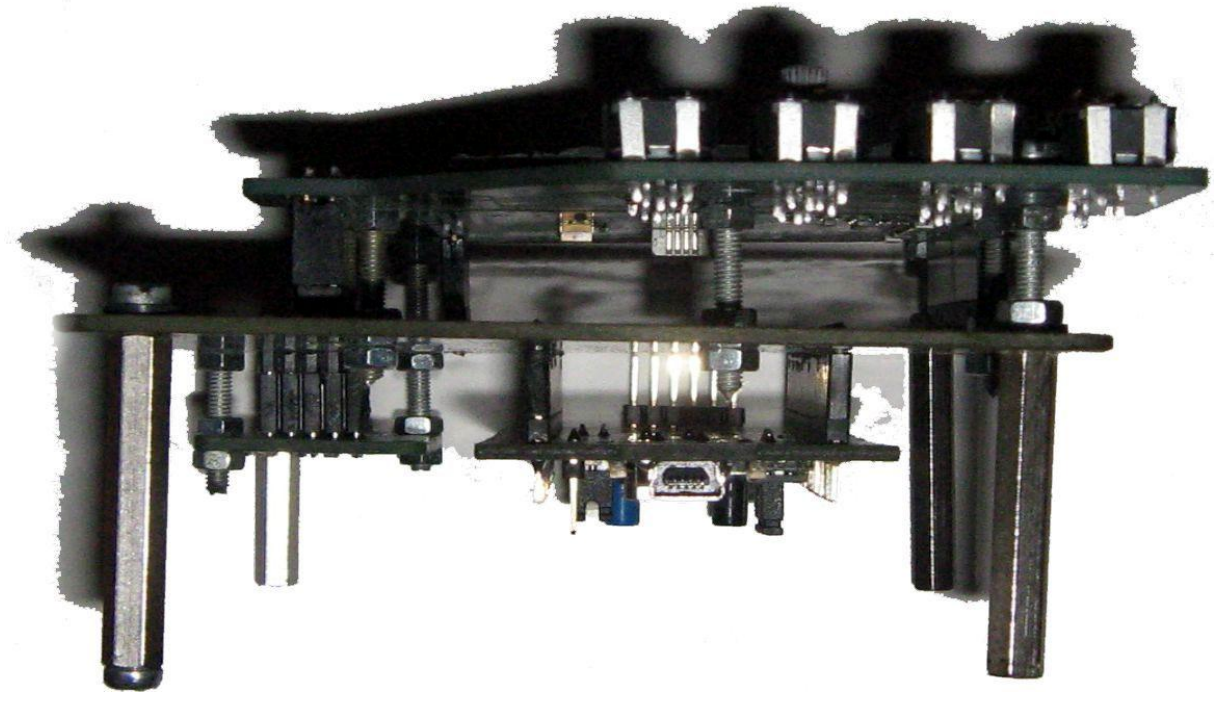

EasyTFT
POTENTIOMETER
BOARD
HW REV. 1.00
Pantelija Brajic 12/3004
MSIBMP 2012/2013


POTENTIOMETER
BOARD


POTENTIOMETER
BOARD
HW REV. 1.00
EasyTFT
262K COLORS
320X240
MSIBMP 2012/2013
Poncelija Brojic 12/3044







Ова страница је намерно остављена празна

Закључак

У овом извештају пројектног задатка изложено је једно од могућих решења датог проблема. Аутор се пре свега трудио да задовољи спецификације пројектног задратка. Решење није најоптималније и оставња доста простора за унапређивање и проширивање могућности. Из претходно изложеног може се видети један класичан пример рада са тасковима. Очигледне су предности које оперативни систем пружа са својим *API* функцијама, које се испољавају у једноставности писања кода, доброј прегледности, а самим тим и бољој контроли над целим системом. Такође, може се приметити и једна очигледна мана везана за коришћење готових *API* функција, која се одоси на време извршавања које може бити доста дуго.

У пројекту није размазтан режим мале потрошње процесора и *TFT* дисплеја. Могуће је угасити дисплеј након неког времена ако је аквизиција заустављена и процесор ставити у режим мале потрошње. Излазак из тог режима може се постићи екстерним прекидом *EXTI*9_5.

Ова страница је намерно остављена празна

4. Додатак

4.1. Кôд главног програма

```
/*******************************************************************************
  * @file    main.c
  * @author  Pantelija Brajic
  * @version V1.0.0
  * @date    1/20/2012
  * @brief   Main program body.
*******************************************************************************/

#include "stm32f10x.h"  // CMSIS Cortex-M3 device peripheral access layer header file
#include "misc.h"       // NVIC and SysTick access layer header file
#include "rtos.h"       // embOS operating system header file

#define ADC1_DR_Address ((u32) 0x4001244C)  // Memmory address of ADC_DR register
#define XY_SAMPLES (8+1)                    // Number of touch-screen samples per axis plus one for averaging
#define EVENT_DMATC (1)                     // DMA transfer complited
#define EVENT_BPWM (2)                      // Sampling frequency changed evet
#define EVENT_START (4)                     // TIM1 enabled event
#define EVENT_STOP (8)                      // TIM1 disabled evet
#define EVENT_FREQ (16)                     // Sampling frequency changed event

OS_STACKPTR int tft_stack[256], pen_stack[256];  // Task stacks
OS_TASK TFT_TASK, PEN_TASK;                      // Task control blocks
OS_CSEMA PEN_CSEMA;                              // Counting semaphore for pen_task synchonization

GPIO_InitTypeDef        GPIO_InitStructure;         // GPIO initialization structure declration
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;  // Timer time base initialization structure declaration
TIM_OCInitTypeDef       TIM_OCInitStructure;        // Timer output compare initialization structure declaration
ADC_InitTypeDef         ADC_InitStructure;          // ADC initialization structure declaration
EXTI_InitTypeDef        EXTI_InitStructure;         // External interrupt initialization structure declaration
NVIC_InitTypeDef        NVIC_InitStructure;         // NVIC initialization structure declaration
DMA_InitTypeDef         DMA_InitStructure;          // DMA initialization structure declaration

char character[60][12] = {{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Character space
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x08, 0x08, 0x08, 0x7F, 0x08, 0x08, 0x08, 0x00, 0x00, 0x00},   // Character +
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x08, 0x30, 0x00},   // Character ,
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Character -
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00},   // Character .
                          {0x02, 0x04, 0x04, 0x0C, 0x08, 0x18, 0x10, 0x10, 0x20, 0x20, 0x40, 0x00},   // Character /
                          {0x00, 0x3c, 0x24, 0x42, 0x46, 0x5A, 0x62, 0x42, 0x24, 0x3C, 0x00, 0x00},   // Character 0
                          {0x00, 0x08, 0x18, 0x28, 0x08, 0x08, 0x08, 0x08, 0x08, 0x3E, 0x00, 0x00},   // Character 1
                          {0x00, 0x3C, 0x42, 0x02, 0x02, 0x04, 0x0C, 0x18, 0x30, 0x7E, 0x00, 0x00},   // Character 2
                          {0x00, 0x3C, 0x02, 0x02, 0x02, 0x1C, 0x02, 0x02, 0x02, 0x3C, 0x00, 0x00},   // Character 3
                          {0x00, 0x0C, 0x0C, 0x14, 0x34, 0x24, 0x44, 0x7F, 0x04, 0x04, 0x00, 0x00},   // Character 4
                          {0x00, 0x3E, 0x20, 0x20, 0x20, 0x3C, 0x02, 0x02, 0x06, 0x3C, 0x00, 0x00},   // Character 5
                          {0x00, 0x1C, 0x20, 0x40, 0x5C, 0x62, 0x42, 0x42, 0x66, 0x3C, 0x00, 0x00},   // Character 6
                          {0x00, 0x7E, 0x02, 0x04, 0x04, 0x08, 0x08, 0x10, 0x10, 0x30, 0x00, 0x00},   // Character 7
                          {0x00, 0x3C, 0x42, 0x42, 0x66, 0x18, 0x66, 0x42, 0x42, 0x3C, 0x00, 0x00},   // Character 8
                          {0x00, 0x3C, 0x66, 0x42, 0x42, 0x46, 0x3A, 0x02, 0x04, 0x38, 0x00, 0x00},   // Character 9
                          {0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00},   // Character :
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                          {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
```

```
                        {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},   // Undefined character
                        {0x00, 0x0C, 0x14, 0x14, 0x36, 0x22, 0x22, 0x7F, 0x41, 0x41, 0x00, 0x00},   // Character A
                        {0x00, 0x7C, 0x42, 0x42, 0x46, 0x7C, 0x42, 0x42, 0x42, 0x7C, 0x00, 0x00},   // Character B
                        {0x00, 0x1C, 0x22, 0x40, 0x40, 0x40, 0x40, 0x40, 0x22, 0x1C, 0x00, 0x00},   // Character C
                        {0x00, 0x78, 0x44, 0x42, 0x42, 0x42, 0x42, 0x42, 0x44, 0x78, 0x00, 0x00},   // Character D
                        {0x00, 0x3E, 0x20, 0x20, 0x20, 0x3E, 0x20, 0x20, 0x20, 0x3E, 0x00, 0x00},   // Character E
                        {0x00, 0x3E, 0x20, 0x20, 0x20, 0x3E, 0x20, 0x20, 0x20, 0x20, 0x00, 0x00},   // Character F
                        {0x00, 0x1C, 0x22, 0x40, 0x40, 0x4E, 0x42, 0x42, 0x22, 0x1E, 0x00, 0x00},   // Character G
                        {0x00, 0x42, 0x42, 0x42, 0x42, 0x7E, 0x42, 0x42, 0x42, 0x42, 0x00, 0x00},   // Character H
                        {0x00, 0x3E, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x3E, 0x00, 0x00},   // Character I
                        {0x00, 0x3E, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x22, 0x1C, 0x00, 0x00},   // Character J
                        {0x00, 0x42, 0x44, 0x48, 0x50, 0x60, 0x50, 0x48, 0x44, 0x42, 0x00, 0x00},   // Character K
                        {0x00, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x3E, 0x00, 0x00},   // Character L
                        {0x00, 0x26, 0x66, 0x5A, 0x5A, 0x5A, 0x52, 0x42, 0x42, 0x42, 0x00, 0x00},   // Character M
                        {0x00, 0x62, 0x62, 0x52, 0x52, 0x52, 0x4A, 0x4A, 0x46, 0x46, 0x00, 0x00},   // Character N
                        {0x00, 0x1C, 0x22, 0x41, 0x41, 0x41, 0x41, 0x41, 0x22, 0x1C, 0x00, 0x00},   // Character O
                        {0x00, 0x7C, 0x46, 0x42, 0x42, 0x46, 0x78, 0x40, 0x40, 0x40, 0x00, 0x00},   // Character P
                        {0x00, 0x1C, 0x22, 0x41, 0x41, 0x41, 0x41, 0x41, 0x22, 0x3C, 0x08, 0x07},   // Character Q
                        {0x00, 0x78, 0x44, 0x44, 0x44, 0x78, 0x48, 0x4C, 0x44, 0x46, 0x00, 0x00},   // Character R
                        {0x00, 0x3C, 0x40, 0x40, 0x60, 0x1C, 0x06, 0x02, 0x02, 0x7C, 0x00, 0x00},   // Character S
                        {0x00, 0x7F, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x00, 0x00},   // Character T
                        {0x00, 0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x46, 0x3C, 0x00, 0x00},   // Character U
                        {0x00, 0x41, 0x41, 0x63, 0x22, 0x22, 0x26, 0x14, 0x14, 0x18, 0x00, 0x00},   // Character V
                        {0x00, 0x41, 0x41, 0x41, 0x59, 0x6B, 0x2A, 0x3A, 0x36, 0x36, 0x00, 0x00},   // Character W
                        {0x00, 0xC6, 0x64, 0x28, 0x18, 0x18, 0x38, 0x24, 0x46, 0xC3, 0x00, 0x00},   // Character X
                        {0x00, 0x41, 0x22, 0x22, 0x14, 0x14, 0x08, 0x08, 0x08, 0x08, 0x00, 0x00},   // Character Y
                        {0x00, 0x3E, 0x04, 0x04, 0x08, 0x18, 0x10, 0x20, 0x20, 0x7E, 0x00, 0x00}};  // Character Z

unsigned x_values[XY_SAMPLES];        // Horisontal touch-screen sample buffer
unsigned y_values[XY_SAMPLES];        // Vertcal touch-screen sample buffer
unsigned short adc_result[8];         // Destination data address of DMA
unsigned bpwm_dr = 10000;             // Back-light duty ratio 50 %
unsigned fsample = 10;                // Sampling frequency multiplied by 10
typedef enum {COMMAND, DATA} RegSel;  // TFT command/data select line values

void DMA1_Channel1_IRQHandler()  // Interrupt service routine for DMA_TC_IRQ
{
  OS_EnterInterrupt();                        // Inform OS that interrupt follows
  if (DMA_GetITStatus(DMA1_IT_TC1) != RESET)  // Check if interrupt occured
  {
    DMA_ClearITPendingBit(DMA1_IT_TC1);      // Acknowledge interrupt
    OS_SignalEvent(EVENT_DMATC, &TFT_TASK);  // Signal that sample data is updated
    OS_LeaveInterrupt();                     // Inform OS that is end of interrupt
  }
}

void EXTI9_5_IRQHandler() // Interrupt service routine for PEN_IRQ
{
  OS_EnterInterrupt();                        // Inform OS that interrupt follows
  if (EXTI_GetITStatus(EXTI_Line6) != RESET)  // Check if interrupt occured
  {
    EXTI_ClearITPendingBit(EXTI_Line6);  // Acknowledge interrupt
    OS_SignalCSema(&PEN_CSEMA);          // Increment counting semaphore PEN_CSEMA
    OS_LeaveInterrupt();                 // Inform OS that is end of interrupt
  }
}

unsigned touch_detect()  // Detect if screen is touched
{
  GPIO_ResetBits(GPIOA, GPIO_Pin_4 | GPIO_Pin_7);          // PA4=0, PA7=0
  OS_Delay(3);                                             // Setup time for PA6 input
  return GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_6) & TRUE;  // Return PA6 state
}

void read_y()  // Sample vertical axis of screen
{
  unsigned i;                                                                  // Temporary counter
  GPIO_ResetBits(GPIOA, GPIO_Pin_4);                                           // PA4=0
  GPIO_SetBits(GPIOA, GPIO_Pin_7);                                             // PA7=1
  ADC_InjectedChannelConfig(ADC1, ADC_Channel_6, 1, ADC_SampleTime_1Cycles5);  // Select PA6 as injected analog input
  OS_Delay(3);                                                                 // Wait for capacitor C1 to charge
  for (i=1; i<XY_SAMPLES; i++)
  {
    ADC_SoftwareStartInjectedConvCmd(ADC1, ENABLE);                             // Capture vertical positin
    while(ADC_GetFlagStatus(ADC1, ADC_FLAG_JEOC) == RESET);                     // Wait for end of injected conversion
    y_values[i] = ADC_GetInjectedConversionValue(ADC1, ADC_InjectedChannel_1);  // Store sample in buffer
  }
}

void read_x()  // Sample horisontal axis of screen
{
  unsigned i;                                                                  // Temporary counter
  GPIO_ResetBits(GPIOA, GPIO_Pin_7);                                           // PA7=0
  GPIO_SetBits(GPIOA, GPIO_Pin_4);                                             // PA4=1
  ADC_InjectedChannelConfig(ADC1, ADC_Channel_5, 1, ADC_SampleTime_1Cycles5);  // Select PA5 as injected analog input
  OS_Delay(3);                                                                 // Wait for capacitor C2 to charge
  for (i=1; i<XY_SAMPLES; i++)
  {
    ADC_SoftwareStartInjectedConvCmd(ADC1, ENABLE);                             // Capture horisontal positin
    while(ADC_GetFlagStatus(ADC1, ADC_FLAG_JEOC) == RESET);                     // Wait for end of injected conversion
    x_values[i] = ADC_GetInjectedConversionValue(ADC1, ADC_InjectedChannel_1);  // Store sample in buffer
  }
}

void tft_write(unsigned regsel, unsigned char code)
```

```
{
  if (regsel == COMMAND)
    GPIO_ResetBits(GPIOA, GPIO_Pin_11);                              // RS=0
  else
    GPIO_SetBits(GPIOA, GPIO_Pin_11);                                // RS=1
  GPIO_ResetBits(GPIOA, GPIO_Pin_13);                                // CS=0
  GPIO_ResetBits(GPIOA, GPIO_Pin_9);                                 // WR=0
  GPIO_ResetBits(GPIOB, GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 |
  GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7);  // Reset pins PB[7:0]
  GPIO_SetBits(GPIOB, code);                                         // Put code on PB[7:0]
  GPIO_SetBits(GPIOA, GPIO_Pin_9);                                   // WR=1
  GPIO_SetBits(GPIOA, GPIO_Pin_13);                                  // CS=1
  GPIO_SetBits(GPIOA, GPIO_Pin_11);                                  // RS=1
}

void tft_set_pos_char(unsigned short x, unsigned short y)  // Set TFT cursor position
{
  char xhs = (x&0xFF00) >> 8;  // Derive higher byte
  char xls = x & 0x00FF;       // Derive lower byte
  x += 7;                      // Increase by symbol width
  char xhe = (x&0xFF00) >> 8;  // Derive higher byte
  char xle = x & 0x00FF;       // Derive lower byte
  char yhs = (y&0xFF00) >> 8;  // Derive higher byte
  char yls = y & 0x00FF;       // Derive lower byte
  y += 11;                     // Increase by symbol height
  char yhe = (y&0xFF00) >> 8;  // Derive higher byte
  char yle = y & 0x00FF;       // Derive lower byte

  tft_write(COMMAND, 0x02);  // Column address start 2
  tft_write(DATA, xhs);      // Higher byte
  tft_write(COMMAND, 0x03);  // Column address start 1
  tft_write(DATA, xls);      // Lower byte
  tft_write(COMMAND, 0x04);  // Column address end 2
  tft_write(DATA, xhe);      // Higher byte
  tft_write(COMMAND, 0x05);  // Column address end 1
  tft_write(DATA, xle);      // Lower byte
  tft_write(COMMAND, 0x06);  // Column address start 2
  tft_write(DATA, yhs);      // Higher byte
  tft_write(COMMAND, 0x07);  // Column address start 1
  tft_write(DATA, yls);      // Lower byte
  tft_write(COMMAND, 0x08);  // Column address end 2
  tft_write(DATA, yhe);      // Higher byte
  tft_write(COMMAND, 0x09);  // Column address end 1
  tft_write(DATA, yle);      // Lower byte
}

void tft_write_char(char c)  // Write number to TFT display
{
  char t;                       // Temporary value
  tft_write(COMMAND, 0x22);     // Write data to GRAM
  for (int i=0; i<12; i++)
  {
    t = character[c-32][i];
    for (int k=0; k<4; k++)
    {
      switch (t&0xC0) {
      case 0x00:                // Both pixels are white
        tft_write(DATA, 0xFF);  // Write R and G of first pixel
        tft_write(DATA, 0xFF);  // Write B of first and B of second pixel
        tft_write(DATA, 0xFF);  // Write G and B of second pixel
        break;
      case 0x40:                // First pixel white second pixel black
        tft_write(DATA, 0xFF);  // Write R and G of first pixel
        tft_write(DATA, 0xF0);  // Write B of first and B of second pixel
        tft_write(DATA, 0x00);  // Write G and B of second pixel
        break;
      case 0x80:                // First pixel black second pixel white
        tft_write(DATA, 0x00);  // Write R and G of first pixel
        tft_write(DATA, 0x0F);  // Write B of first and B of second pixel
        tft_write(DATA, 0xFF);  // Write G and B of second pixel
        break;
      case 0xC0:                // Both pixels black
        tft_write(DATA, 0x00);  // Write R and G of first pixel
        tft_write(DATA, 0x00);  // Write B of first and B of second pixel
        tft_write(DATA, 0x00);  // Write G and B of second pixel
        break;
      }
    t <<= 2;                    // Check following two pixels
    }
  }
}

void tft_write_string(unsigned short x, unsigned short y, char* string)  // Write string to TFT display
{
  int i;                         // Temporary counter
  for (i=0; *string!='\0'; i++)
  {
    tft_set_pos_char(x+8*i, y);
    tft_write_char(*string);
    string++;
  }
}

void pen_task()  // Scan touch position
{
```

```
  unsigned i;  // Temporary counter
  while (1)
  {
    OS_WaitCSema(&PEN_CSEMA);  // Wait for counting semaphore PEN_CSEMA
    OS_Delay(3);               // Debounce the touch
    read_y();  // Sample vertical axis of screen
    read_x();  // Sample horisontal axis of screen

    y_values[0] = 0;               // Reset vertical average value
    for (i=1; i<XY_SAMPLES; i++)   // Sum of vertical samples in average value
    {
      y_values[0] += y_values[i];
    }
    y_values[0] /= XY_SAMPLES-1;   // Divide sum with number of samples
    x_values[0] = 0;               // Reset horisontal average value
    for (i=1; i<XY_SAMPLES; i++)   // Sum of horisontal samples in average value
    {
      x_values[0] += x_values[i];
    }
    x_values[0] /= XY_SAMPLES-1;   // Divide sum with number of samples
    if (touch_detect())            // If still have touch
    {
      if (y_values[0]>1732 && y_values[0]<2207 && x_values[0]>2086 && x_values[0]<2586)
      {
        if (bpwm_dr <= 18000)
        {
          bpwm_dr += 2000;                          // Increment by 10% of full scale
          TIM_UpdateDisableConfig(TIM1, ENABLE);    // Update event disabled
          TIM_SetCompare1(TIM17, bpwm_dr);          // Set new value in capture compare register
          TIM_UpdateDisableConfig(TIM17, DISABLE);  // Update event enabled
          OS_SignalEvent(EVENT_BPWM, &TFT_TASK);    // Signal that sampling frequency is updated
        }
      }
      if (y_values[0]>1732 && y_values[0]<2207 && x_values[0]>2586 && x_values[0]<3086)
      {
        if (bpwm_dr >= 4000)
        {
          bpwm_dr -= 2000;                          // Decrement by 10% of full scale
          TIM_UpdateDisableConfig(TIM17, ENABLE);   // Update event disabled
          TIM_SetCompare1(TIM17, bpwm_dr);          // Set new value in capture compare register
          TIM_UpdateDisableConfig(TIM17, DISABLE);  // Update event enabled
          OS_SignalEvent(EVENT_BPWM, &TFT_TASK);    // Signal that sampling frequency is updated
        }
      }
      if (y_values[0]>1260 && y_values[0]<1732 && x_values[0]>640 && x_values[0]<1976)
      {
        TIM_Cmd(TIM1, ENABLE);                   // Enable counter TIM1 CEN=1
        OS_SignalEvent(EVENT_START, &TFT_TASK);  // Signal that sampling is enabled
      }
      if (y_values[0]>1260 && y_values[0]<1732 && x_values[0]>1976 && x_values[0]<3237)
      {
        TIM_Cmd(TIM1, DISABLE);                  // Disable counter TIM1 CEN=0
        OS_SignalEvent(EVENT_STOP, &TFT_TASK);   // Signal that sampling is disabled
      }
      if (y_values[0]>2207 && y_values[0]<2707 && x_values[0]>2586 && x_values[0]<3086)
      {
        if (fsample >= 10)
        {
          fsample -= 5;                             // Decrement Fs by 0.5 Hz
          TIM_UpdateDisableConfig(TIM1, ENABLE);    // Update event disabled
          TIM_SetAutoreload(TIM1, 40000/fsample);   // Set new value in capture compare register
          TIM_UpdateDisableConfig(TIM1, DISABLE);   // Update event enabled
          OS_SignalEvent(EVENT_FREQ, &TFT_TASK);    // Signal that sampling frequency is updated
        }
      }
      if (y_values[0]>2207 && y_values[0]<2707 && x_values[0]>2086 && x_values[0]<2586)
      {
        if (fsample <= 90)
        {
          fsample += 5;                             // Increment Fs by 0.5 Hz
          TIM_UpdateDisableConfig(TIM1, ENABLE);    // Update event disabled
          TIM_SetAutoreload(TIM1, 40000/fsample);   // Set new value in capture compare register
          TIM_UpdateDisableConfig(TIM1, DISABLE);   // Update event enabled
          OS_SignalEvent(EVENT_FREQ, &TFT_TASK);    // Signal that sampling frequency is updated
        }
      }
    }
    OS_Delay(200);  // Task period 200 ms
  }
}

void tft_task()  // Write data to TFT display
{
  OS_U8 event;  // Event register
  while (1)
  {
    event = OS_WaitEvent(EVENT_DMATC | EVENT_BPWM | EVENT_START | EVENT_STOP | EVENT_FREQ);
    if (event & EVENT_DMATC)
    {
      int hto[3];                                  // Buffer for hundreds, tens and ones
      int i, j, k;                                 // Temporary counters
      for (i=0; i<2; i++)
        for (j=0; j<4; j++)
        {
          if (hto[0] = adc_result[4*i+j]>=4054)    // Get number of hundreds
```

```
          {
            hto[1] = 0;
            hto[2] = 0;
          }
          else
          {
            hto[1] = adc_result[4*i+j] / 410;         // Get number of tens
            hto[2] = (adc_result[4*i+j] % 410) / 41;  // Get number of ones
          }
          for (k=0; k<3; k++)                         // Print to TFT display
          {

            tft_set_pos_char(29+53*j+8*k, 30+42*i);
            tft_write_char(hto[k]+48);
          }
        }
    }
    if (event & EVENT_BPWM)
    {
      int hto[3];                          // Buffer for hundreds, tens and ones
      int i;                               // Temporary counter
      if (hto[0] = bpwm_dr>=19800)
      {
        hto[1] = 0;
        hto[2] = 0;
      }
      else
      {
        hto[1] = bpwm_dr / 2000;           // Get number of tens
        hto[2] = (bpwm_dr % 2000) / 200;   // Get number of ones
      }
      for (i=0; i<3;i++)                   // Print to TFT display
      {
        tft_set_pos_char(188+8*i, 156);
        tft_write_char(hto[i]+48);
      }
    }
    if (event & EVENT_START)
    {
      tft_write_string(53, 198, "     ");
      tft_write_string(146, 198, "STOP");
    }
    if (event & EVENT_STOP)
    {
      tft_write_string(53, 198, "START");
      tft_write_string(146, 198, "    ");
    }
    if (event & EVENT_FREQ)
    {
      int hto[3];                          // Buffer for hundreds, tens and ones
      int i;                               // Temporary counter
      hto[0] = fsample/10 + 2;             // Get number of tens
      hto[1] = 0;
      hto[2] = fsample%10 + 2;             // Get number of ones
      for (i=0; i<3;i++)                   // Print to TFT display
      {
        tft_set_pos_char(188+8*i, 114);
        tft_write_char(hto[i]+46);
      }
    }
  }
}

void rcc_init()
{
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);  // Clock source to GPIOA
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);  // Clock source to GPIOB
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);   // Enable AFIO clock source
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);   // Enable ADC1 clock source
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);   // Enable TIM1 clock source
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM17, ENABLE);  // Enable TIM17 clock source
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);     // Enable DMA clock source
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);  // Clock source to GPIOC
}

void gpio_init()
{
  GPIO_PinRemapConfig(GPIO_Remap_SWJ_Disable, ENABLE);     // Remap PA13 to alternate function
  GPIO_StructInit(&GPIO_InitStructure);                    // Reset initialization structure
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;         // Output speed 2 MHz
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;         // Output push-pull
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_7 |
  GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_11 | GPIO_Pin_13;    // Select pins PA4, PA7, PA8, PA9, PA11, PA13
  GPIO_Init(GPIOA, &GPIO_InitStructure);                   // GPIOA port initialization
  GPIO_ResetBits(GPIOA, GPIO_Pin_4 | GPIO_Pin_7);          // PA4=0, PA7=0
  GPIO_SetBits(GPIOA, GPIO_Pin_8 | GPIO_Pin_9 |
               GPIO_Pin_11 | GPIO_Pin_13);                 // CS=1, RS=1, RD=1, WR=1
  GPIO_StructInit(&GPIO_InitStructure);                    // Reset initialization structure
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;            // Analog in
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 |  // Select pins PA0, PA1, PA2, PA3, PA5
  GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_5;                    // PA6 floatin input because of external interrupt
  GPIO_Init(GPIOA, &GPIO_InitStructure);                   // GPIOA port initialization

  GPIO_StructInit(&GPIO_InitStructure);                    // Reset initialization structure
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;         // Output speed 2 MHz
```

```
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;          // Alternative function open drain
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;                // Select PB9
  GPIO_Init(GPIOB, &GPIO_InitStructure);                   // GPIOB port initialization
  GPIO_StructInit(&GPIO_InitStructure);                    // Reset initialization structure
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;         // General purpose push-pull
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 |
  GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
  GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8;                    // Select pins PB[8:0]
  GPIO_Init(GPIOB, &GPIO_InitStructure);                   // GPIOB port initialization
  GPIO_ResetBits(GPIOB, GPIO_Pin_All);
  GPIO_SetBits(GPIOB, GPIO_Pin_8);                         // RST=1, no display reset

  GPIO_StructInit(&GPIO_InitStructure);                    // Reset initialization structure
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;            // Analog in
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 |
  GPIO_Pin_2 | GPIO_Pin_3;                                 // Select pins PC[3:0]
  GPIO_Init(GPIOC, &GPIO_InitStructure);                   // GPIOC port initialization
}

void adc_init()
{
  ADC_StructInit(&ADC_InitStructure);                                          // Reset ADC init structure
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;        // TIM4_CH4 trigger conversion
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;                           // Independent operation mode
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;                          // Continuous conversion mode enabled
  ADC_InitStructure.ADC_ScanConvMode = ENABLE;                                 // Scan conversion mode enabled
  ADC_InitStructure.ADC_NbrOfChannel = 8;                                      // Number of ADC active channels
  ADC_Init(ADC1, &ADC_InitStructure);                                          // ADC initialization structure
  ADC_RegularChannelConfig(ADC1, ADC_Channel_13, 1, ADC_SampleTime_1Cycles5);  // Select PC3 as 1st conversion
  ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 2, ADC_SampleTime_1Cycles5);   // Select PA0 as 2nd conversion
  ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 3, ADC_SampleTime_1Cycles5);  // Select PC0 as 3rd conversion
  ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 4, ADC_SampleTime_1Cycles5);   // Select PA1 as 4th conversion
  ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 5, ADC_SampleTime_1Cycles5);  // Select PC1 as 5th conversion
  ADC_RegularChannelConfig(ADC1, ADC_Channel_2, 6, ADC_SampleTime_1Cycles5);   // Select PA2 as 6th conversion
  ADC_RegularChannelConfig(ADC1, ADC_Channel_12, 7, ADC_SampleTime_1Cycles5);  // Select PC2 as 7th conversion
  ADC_RegularChannelConfig(ADC1, ADC_Channel_3, 8, ADC_SampleTime_1Cycles5);   // Select PA3 as 8th conversion
  ADC_ExternalTrigConvCmd(ADC1, ENABLE);                                       // Enable external trigerrig mode
  ADC_DMACmd(ADC1, ENABLE);                                                    // Enable DMA data transfer for ADC
  ADC_ExternalTrigInjectedConvConfig(ADC1, ADC_ExternalTrigInjecConv_None);    // Software trigger injected conversion
  ADC_ExternalTrigInjectedConvCmd(ADC1, ENABLE);                               // Enable external injected triggering mode
  ADC_Cmd(ADC1, ENABLE);                                                       // Turn ADC on
  ADC_ResetCalibration(ADC1);                                                  // Enable ADC1 reset calibaration registers
  while (ADC_GetResetCalibrationStatus(ADC1));                                 // Check the end of ADC1 reset calibration
  ADC_StartCalibration(ADC1);                                                  // Start ADC calibaration
  while(ADC_GetCalibrationStatus(ADC1));                                       // Wait for end of ADC calibration
}

void tim_init(void)
{
  TIM_TimeBaseStructInit(&TIM_TimeBaseInitStructure);              // Reset time base init structe
  TIM_TimeBaseInitStructure.TIM_Prescaler = 12;                    // Clock prescaler value Fclk = 1MHz
  TIM_TimeBaseInitStructure.TIM_Period = 20000;                    // Auto-relaod register value Fcnt = 1000 Hz
  TIM_TimeBaseInit(TIM17, &TIM_TimeBaseInitStructure);             // TIM17 time base initialization
  TIM_OCStructInit(&TIM_OCInitStructure);                          // Reset output compare init structure
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;    // Enable capture compare output CC1E=1
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;                // PWM mode 2 output compare mode
  TIM_OCInitStructure.TIM_Pulse = bpwm_dr;                         // Capture compare register value (BPWM duty ratio)
  TIM_OC1Init(TIM17, &TIM_OCInitStructure);                        // TIM17 output compare initialization
  TIM_ARRPreloadConfig(TIM17, ENABLE);                             // Enable auto-reload preload ARPE=1
  TIM_Cmd(TIM17, ENABLE);                                          // Enable counter CEN=1
  TIM_CtrlPWMOutputs(TIM17, ENABLE);                               // Enable main Outputs MOE=1
  TIM_TimeBaseStructInit(&TIM_TimeBaseInitStructure);              // Reset time base init structe
  TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;  // Set TIM1 counter up
  TIM_TimeBaseInitStructure.TIM_Prescaler = 6000;                  // Clock prescaler value
  TIM_TimeBaseInitStructure.TIM_Period = 40000/fsample;            // Auto-relaod register value Fs = 1Hz
  TIM_TimeBaseInit(TIM1, &TIM_TimeBaseInitStructure);              // TIM1 time base initialization

  TIM_OCStructInit(&TIM_OCInitStructure);                         // Reset output compare init structure
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;   // Enable capture compare output CC1E=1
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;               // PWM mode 2 output compare mode
  TIM_OCInitStructure.TIM_Pulse = 400;                            // Capture compare register value
  TIM_OC1Init(TIM1, &TIM_OCInitStructure);                        // TIM17 output compare initialization
  TIM_ARRPreloadConfig(TIM1, ENABLE);                             // Enable auto-reload preload ARPE=1
  TIM_Cmd(TIM1, DISABLE);                                         // Disable counter CEN=0
  TIM_CtrlPWMOutputs(TIM1, ENABLE);                               // Enable PWM outputs on TIM1
}

void nvic_init()
{
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);                  // 3 bits for pre-emption priority, 1 bits for subpriority
  NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;               // Enable EXTI line[9:5] interrupts
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0xF0>>5;  // Preemt priority level 240
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;               // Subpriority level 1
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;                  // Enable EXTI9_5_IRQn channel
  NVIC_Init(&NVIC_InitStructure);                                  // Initialize NVIC
  NVIC_InitStructure.NVIC_IRQChannel = DMA1_Channel1_IRQn;         // Enable EXTI line[9:5] interrupts
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0xE0>>5;  // Preemt priority level 224
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;               // Subpriority level 1
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;                  // Enable EXTI9_5_IRQn channel
  NVIC_Init(&NVIC_InitStructure);
}

void exti_init()
{
  EXTI_StructInit(&EXTI_InitStructure);                       // Reset EXTI init structure
```

```
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;                    // Enable external interrupt
  EXTI_InitStructure.EXTI_Line = EXTI_Line6;                   // Map Px5 lines as external interrupt
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;          // Interrupt mode for Px6 lines
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;       // Rising edge triggering
  EXTI_Init(&EXTI_InitStructure);                              // Initialize EXTI
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource6);  // PA6 used as EXTI line
}

void dma_init()
{
  DMA_StructInit(&DMA_InitStructure);                                          // Reset DMA init structure
  DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;                  // ADC_DR register address for source
  DMA_InitStructure.DMA_MemoryBaseAddr = (vu32) adc_result;                    // Memory destination address
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;                           // Direction from peripheral to memory
  DMA_InitStructure.DMA_BufferSize = 8;                                        // Destination buffer size
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;             // Disable increment source address
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;                      // Enable increment memory address
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;  // Half-word peripheral data width
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;          // Half-word memory data width
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;                              // Enable circular destination buffer
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;                          // Channel priority level
  DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;                                 // Disable memory to memory transfer
  DMA_Init(DMA1_Channel1, &DMA_InitStructure);                                 // Initialize DMA
  DMA_ITConfig(DMA1_Channel1, DMA_IT_TC, ENABLE);                              // Enable tranfer complete interrupt
  DMA_Cmd(DMA1_Channel1, ENABLE);                                              // Enable DMA1 channel 1
}

void tft_init()
{
  GPIO_ResetBits(GPIOB, GPIO_Pin_8);  // RST=0
  for (unsigned i=0; i< 4000; i++);   // Delay 1ms
  GPIO_SetBits(GPIOB, GPIO_Pin_8);    // RST=1
  for (unsigned i=0; i< 20000; i++);  // Delay 5ms

  tft_write(COMMAND, 0x1A);           // Power control 1
  tft_write(DATA, 0x01);              // Set gate output voltage range VGH=15, VGL=-10
  tft_write(COMMAND, 0x1B);           // Power control 2
  tft_write(DATA, 0x1B);              // Gamma voltage adjustment VREG1=4.65
  tft_write(COMMAND, 0x23);           // VCOM control 1
  tft_write(DATA, 0x80);              // VCOM offset voltage 0mV
  tft_write(COMMAND, 0x24);           // VCOM control 2
  tft_write(DATA, 0x71);              // Common electrode output high voltage VCOMH=4.2
  tft_write(COMMAND, 0x25);           // VCOM control 3
  tft_write(DATA, 0x2F);              // Common electrode output low voltage VCOML=-1.8
  tft_write(COMMAND, 0x19);           // OSC control 1
  tft_write(DATA, 0x01);              // Enable internal oscillator OSC_EN=1
  tft_write(COMMAND, 0x1C);           // Power control 3
  tft_write(DATA, 0x03);              // Amount of current driving for OP amplifier small
  tft_write(COMMAND, 0x1F);           // Power control 6
  tft_write(DATA, 0x88);              // Exit standby mode
  tft_write(COMMAND, 0x1F);           // Power control 6
  tft_write(DATA, 0x80);              // Turn on step-up circuit 1
  tft_write(COMMAND, 0x1F);           // Power control 6
  tft_write(DATA, 0x90);              // Turn on step-up circuit 2
  tft_write(COMMAND, 0x1F);           // Power control 6
  tft_write(DATA, 0xD0);              // VCOML can output to negative
  for (unsigned i=0; i< 10000; i++);  // Delay 10ms
  tft_write(COMMAND, 0x28);           // Display control 3
  tft_write(DATA, 0x3C);              // Turn display on
  tft_write(COMMAND, 0x17);           // Colour mode
  tft_write(DATA, 0x03);              // 12 bits per pixel
  tft_write(COMMAND, 0x16);           // Memory access control
  tft_write(DATA, 0x00);              // MX=0, MY=0, MV=0
  tft_write(COMMAND, 0x02);           // Column address start 2
  tft_write(DATA, 0x00);              // 0
  tft_write(COMMAND, 0x03);           // Column address start 1
  tft_write(DATA, 0x00);              // 0
  tft_write(COMMAND, 0x04);           // Column address end 2
  tft_write(DATA, 0x00);              // 0
  tft_write(COMMAND, 0x05);           // Column address end 1
  tft_write(DATA, 0xEF);              // 239
  tft_write(COMMAND, 0x06);           // Row address start 2
  tft_write(DATA, 0x00);              // 0
  tft_write(COMMAND, 0x07);           // Row address start 1
  tft_write(DATA, 0x00);              // 0
  tft_write(COMMAND, 0x08);           // Row address end 2
  tft_write(DATA, 0x01);              // 1
  tft_write(COMMAND, 0x09);           // Row address end 1
  tft_write(DATA, 0x3F);              // 63
  tft_write(COMMAND, 0x22);           // Write data to GRAM
  for (int i=0; i<76800; i++)         // White display
  {
    tft_write(DATA, 0xFF);
    tft_write(DATA, 0xFF);
    tft_write(DATA, 0xFF);
  }
  tft_write_string(188, 114, "1.0");
  tft_write_string(188, 156, "050");
  tft_write_string(44, 114, "FREQUENCY:  +  -");
  tft_write_string(36, 156, "BRIGHTNESS:  +  -");
  tft_write_string(53, 198, "START");
  tft_write_string(104, 296, "MS1BMP 2012/2013.");
  tft_write_string(47, 308, "PANTELIJA BRAJIC 12/3094");

}
```

```
int main(void)
{
  OS_IncDI();      // Initially disable interrupts
  OS_InitKern();   // Initialize embOS
  OS_InitHW();     // Initialize hardware for embOS
  rcc_init();      // Initialize reset and clock controller
  gpio_init();     // Initialize GPIO prots
  adc_init();      // Initialize A/D convertor
  nvic_init();     // Initialize nasted vector interrupt contoller
  exti_init();     // Initialize external interrupts
  dma_init();      // Initialize direct memory access controller
  tim_init();      // Initialize timers TIM1 and TIM17
  tft_init();      // Initialize TFT display

  OS_CREATETASK(&PEN_TASK, "pen_task", pen_task, 100, pen_stack);  // Task for scanning touch-panel
  OS_CREATETASK(&TFT_TASK, "tft_task", tft_task, 50, tft_stack);   // Task for displaying data on TFT
  OS_CREATECSEMA(&PEN_CSEMA);                                      // Semaphore for pen_task sync

  OS_DecRI();  // Enable interupts
  OS_Start();  // Start multitasking
  return 0;
}
```

Литература

[1] *STMicrocontrolers*: "*STM*32*VLDISCOVERY STM*32 *value line Discovery*", http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/USER_MANUAL/CD00267113.pdf.

[2] *STMicrocontrolers*: "*STM*32*F*100*xx advanced ARM-based* 32*-bit MCUs – Reference manual*", http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/REFERENCE_MANUAL/CD00246267.pdf.

[3] *STMicrocontrolers*: "*Low & medium-density value line, advanced ARM-based 32-bit MCU with 16 to 128 KB Flash, 12 timers, ADC, DAC & 8 comm interfaces*", http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00251732.pdf.

[4] *Himax*: "*HX*8347*-D* 240*RGB x* 320 *dot,* 262*K color, with internal GRAM, TFT Mobile Single Chip Driver*", www.displayfuture.com/Display/datasheet/controller/HX8347-D.pdf.

[5] *MULTI-INNO TECHNOLOGY*: "*LCD MODULE SPECIFICATION MI0240ST*-3", http://www.mikroe.com/downloads/get/1009/ili9340_spec.pdf.

[6] *IAR Systems*, http://www.iar.com/.

[7] *SEGGER*, http://www.segger.com/.