Kārlis Raimonds Eniņš, Gints Romanovskis LLU ITF MAG, 1.kurss. 14.03.2020.

## **Darbs tiešsaistē priekšmetā “Datorizētā projektēšana, modelēšana un ražošana”**

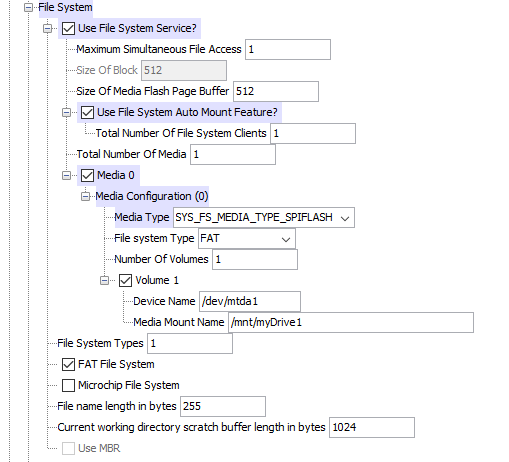
# **CASE rīki iegultajā programmēšanā.**

5.variants: USB flash hosts

Izmantotais mikrokontrolieris: PIC32MX795F512L

## MPLAB Harmony configurator konfigrācija

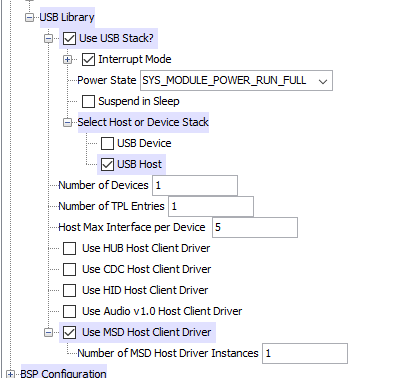
Pie opcijām sadaļā “Harmony Framework Configuration” ->”System Services” -> “File System” jāieslēdz failu sistēmas izmantošana. Jāiestata Failu sistēmas tips uz FAT, kā arī jāpārbauda kāds ir kokrētās ierīces vārds un piemontēšanas vārds.



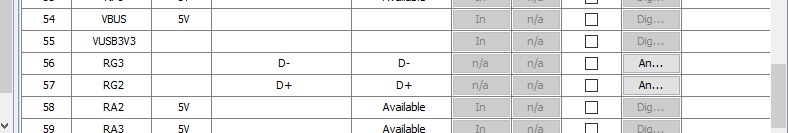
Piemontēšanas vārds vēlāk tiek izmantots kodā, lai norādītu, kur izvietot failu.

appData.fileHandle = SYS\_FS\_FileOpen("/mnt/myDrive1/simpleText.txt",(SYS\_FS\_FILE\_OPEN\_APPEND\_PLUS));

Pie opcijām sadaļā “Harmony Framework Configuration” ->”System Services” -> “USB Library” Jāieslēdz USB Stack, jānomaina tips no Device uz Host Stack un jāiesta MSD (Mass storage device) Host Client driver.



Mikrokontrolierim ar Harmony configurator palīdzību nokonfigurē USB paredzētās datu kājiņas RG2(57) un RG3 (56) kā D+ un D-. Attēlā redzams, ka VBUS(54) un VUSB3V3(55) jau ir pēc noklusējuma nokonfigurētas attiecīgi nepieciešamajām funkcijām.



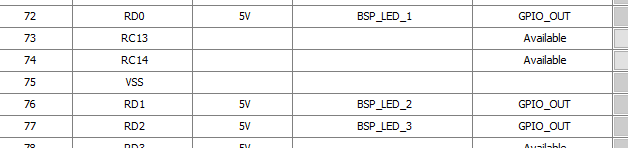
RF3(51) kājiņu kā USB ID



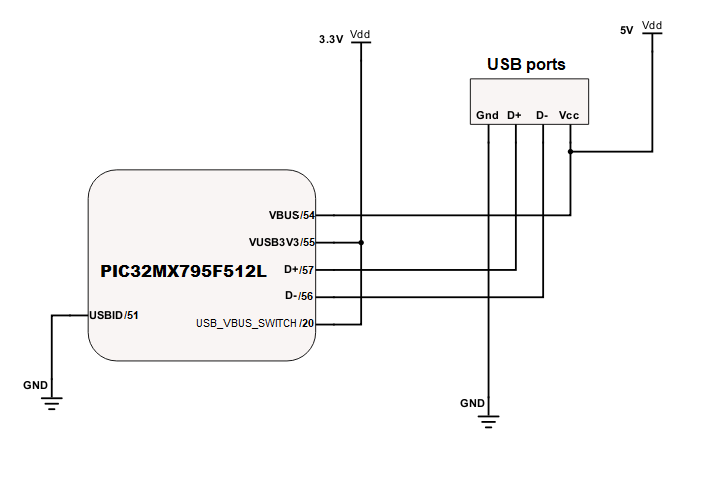
Un RB5(20) kā USB\_VBUS\_SWITCH (VBUS)



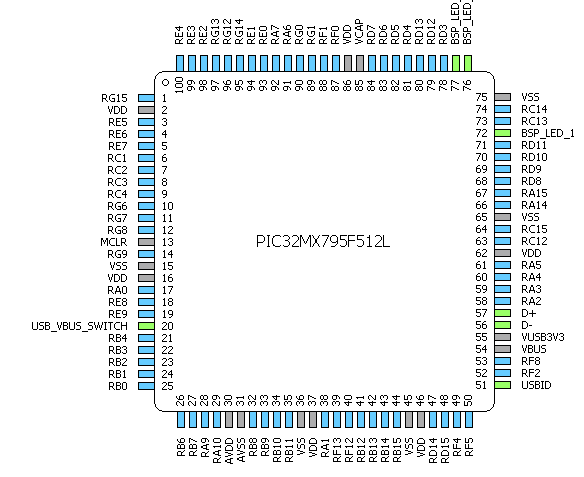
Papildus nokonfigurē visas trīs uz prototipēšanas plates esošās LED, kur attiecīgi BSP\_LED\_1 ir pieslēgts pie RD0(72), BSP\_LED\_2 pie RD1(76) un BSP\_LED\_3 pie RD2(77). Šīs LED tiek izmantotas, piemēram, programmatūras izpildes kļūdas parādīšanai.



USB ports uz prototipēšanas platformas “PIC32 Ethernet Starter Kit” pie mikrokontrolera pieslēgts pēc šādas shēmas.



Un nokonfigurējot pinus, Pin diagram izskatās šādi.



## Pievienotais kods

Papildus, jau klasē izstrādātajai projekta daļai ar 2 mirgojošiem LED, bija jāpievieno kods darbībai ar failsistēmu un ar USB ierīces pieslēgumiem.

Visa programmas darība ir organizēta ar “switch” palīdzība. Katrā “case” tiek organizēta zināmu procedūru izpilde un to izpildes pārbaude. Ja scenārija izpilde ir korekta, notiek pāriešana uz nākamo “case”, ja neizpildās, pāriet uz “Error case”.

Lai saprastu kādā stāvoklī atrodas sistēma, ir sekojoši uz prototipēšanas plates esošo LED kodi:

|  |  |  |
| --- | --- | --- |
| LED numurs | Stāvoklis(On/Off) | Nozīme |
| LED1 | On | Ir pievienota USB ierīce |
| LED2 | On | Gaidīšanas režīms |
| LED3 | On | Kļūda |
| Visi LED | Off |  |

Galvenā funkcija kodā ir APP\_Tasks , kura iekļauj visu Switch konstrukciju.

|  |
| --- |
| **void** **APP\_Tasks** ( **void** ) {  //LED tiek slēgti sekojoshi:    //LED1 - On - Pievienota ieriice  //LED2 - On - Gaidiishanas rezhiims  //LED3 - On - ERROR  //Visi LED - Off - nav piesleegts USB Flash  /\* Check the application's current state. \*/  **switch** ( appData.state )  {  /\* Application's initial state. \*/  **case** APP\_STATE\_INIT:  {  **bool** appInitialized = true;  //izsleedz visus LED  BSP\_LED\_1Off();  BSP\_LED\_2Off();  BSP\_LED\_3Off();    //inicializee event handleri.  SYS\_FS\_EventHandlerSet(APP\_SYSFSEventHandler, (**uintptr\_t**)NULL);  **if** (appInitialized)  {  //pagaidaam sheit lai ir shaadi  appData.state = APP\_STATE\_USB;  }  **break**;  }//inicializee USB hostu  **case** APP\_STATE\_USB:  USB\_HOST\_EventHandlerSet(APP\_USBHostEventHandler, 0);  USB\_HOST\_BusEnable(0);  **if**(USB\_HOST\_BusIsEnabled(0))  {  appData.state = APP\_WAIT\_FOR\_DEVICE\_ATTACH;  }  **break**;  //gaida liidz tiks pievienota ieriice  **case** APP\_WAIT\_FOR\_DEVICE\_ATTACH:   //kad ieriice pievienota, paarsleedzas uz naakamo staavokli APP\_OPEN\_FILE  **if**(appData.deviceIsConnected)  {  BSP\_LED\_1On();//iesleedz 1.LED  BSP\_LED\_2Off();//izsleedz 2.diodi.  appData.state = APP\_OPEN\_FILE;  }   **break**;    //  //sheit naaks viss kas saistiits ar usb pievienoshanu.  //  //atveram failu, kuru rakstiit uz USB. Pagaidaam bez usb.  **case** APP\_OPEN\_FILE:    appData.fileHandle = SYS\_FS\_FileOpen("/mnt/myDrive1/simpleText.txt",(SYS\_FS\_FILE\_OPEN\_APPEND\_PLUS));  **if**(appData.fileHandle == SYS\_FS\_HANDLE\_INVALID)  {  //ja failu nevar atveert , atgriezh kljuudu  appData.state = APP\_ERROR;  }  **else**  {    //ja failu var atveert, paariet uz rakstiishanu  appData.state = APP\_STATE\_WRITE\_TO\_FILE;   }  **break**; //raksta failu uz failsisteemu.  **case** APP\_STATE\_WRITE\_TO\_FILE:     **if** (SYS\_FS\_FileWrite( appData.fileHandle, (**const** **void** \*) writeData, 12 ) == -1)  {  //neizdodas rakstiit uz failu, atgriezh kljuudu  SYS\_FS\_FileClose(appData.fileHandle);  appData.state = APP\_ERROR;   }  **else**  {  //ja izdodads ierakstiit, aizver failu  appData.state = APP\_STATE\_CLOSE\_FILE;  }   **break**; //aizver failu  **case** APP\_STATE\_CLOSE\_FILE:     SYS\_FS\_FileClose(appData.fileHandle);   //paarsleedzas uz IDLE  appData.state = APP\_STATE\_END\_IDLE;  **break**;  //gaidiishanas rezhiims  **case** APP\_STATE\_END\_IDLE:  printf("Success! ");  BSP\_LED\_2On();  BSP\_LED\_1Off();  **if**(appData.deviceIsConnected == false)  {  appData.state = APP\_WAIT\_FOR\_DEVICE\_ATTACH;  BSP\_LED\_2Off();  }  **break**;     **case** APP\_ERROR:  // kljuudas gadiijumaaIesleedz tresho LED uz plates.  BSP\_LED\_3On();  **break**;  /\* The default state should never be executed. \*/  **default**:  {  /\* **TODO:** Handle error in application's state machine. \*/  **break**;  }  } } |

Lai nodrošinātu šīs funkcijas darbību nepieciešamas divas palīgfunkcijas

|  |
| --- |
| //eventhandleris - atkariibaa no staavokla //atziimee vai failsisteemas ieriice ir piemonteeta vai nee. . **void** **APP\_SYSFSEventHandler**(SYS\_FS\_EVENT event, **void** \* eventData, **uintptr\_t** context) {  **switch**(event)  {  **case** SYS\_FS\_EVENT\_MOUNT:  appData.deviceIsConnected = true;  **break**;    **case** SYS\_FS\_EVENT\_UNMOUNT:  appData.deviceIsConnected = false;    **break**;    **default**:  **break**;  } } //usb host eventhandler USB\_HOST\_EVENT\_RESPONSE **APP\_USBHostEventHandler** (USB\_HOST\_EVENT event, **void** \* eventData, **uintptr\_t** context) {  **switch** (event)  {  **case** USB\_HOST\_EVENT\_DEVICE\_UNSUPPORTED:  **break**;  **default**:  **break**;    }    **return**(USB\_HOST\_EVENT\_RESPONSE\_NONE); } |

Kā arī iekš app.h faila ir jānodefinē izmantotie APP\_STATES

|  |
| --- |
| **typedef** **enum** {  /\* Application's state machine's initial state. \*/  APP\_STATE\_INIT=0,  APP\_STATE\_MOUNT\_DISK,  APP\_STATE\_UNMOUNT\_DISK,  APP\_OPEN\_FILE,  APP\_STATE\_WRITE\_TO\_FILE,  APP\_STATE\_CLOSE\_FILE,  APP\_ERROR,  APP\_STATE\_END\_IDLE,  APP\_STATE\_USB,  APP\_WAIT\_FOR\_DEVICE\_ATTACH,  /\* **TODO:** Define states used by the application state machine. \*/ } APP\_STATES; |