ВГРАЖДАНЕ НА CDC ХОСТ ЗА УПРАВЛЕНИЕ НА 2D ЛАЗЕРЕН СКЕНЕР С ПРИЛОЖЕНИЕ В МОБИЛНАТА РОБОТИКА

д-р инж. Атанас Иванов Димитров
Бургаски свободен университет

EMBEDDED CDC HOST FOR CONTROL OF A 2D LASER SCANNER WITH APPLICATION IN MOBILE ROBOTATION

Atanas Ivanov Dimitrov, PhD
Burgas Free University

Abstract: *In this paper, the implementation (hardware and firmware) of on-board USB CDC host, using VNC2 controller for communication with 2D laser scanner URG-04LX-UG01 is presented. The presented system is embedded in small tracked mobile robot with a basic limit for minimum energy consumption. The main accent is on software implementation of CDC host and its operation with the real time operating system without this leading to increasing normal error of measurements of the used sensor.*

Key words: *Embedded CDC Host, VNC2, LIDAR, URG-04LX-UG01*

Въведение

Мобилните роботи възприемат заобикалящия ги свят посредством изградената им бордова сензорна система. За оператора, водещ робота, е от съществено значение да получава непрекъснато достоверна и лесна за възприемане информация за наличието, разстоянието и разположението на различни препятствия и обекти в работната сцена.

За измерване на средни разстояния в околност около роботите се използват предимно пиезоелектрични ултразвукови сензори с обхват от 1 ÷ 7 m. В диапазона от 10 ÷ 80 cm се използват инфрачервени (IR) сензори, разположени най-често в основата на мобилните платформи (МП) [1] – [4], [10, 11].

Малката разделителна способност на по-горе споменатите сензори не осигурява достатъчно детайлна информация за формата, големината и ъгъла, под който са разположени препятствията спрямо МП на робота, което възпрепятства съответното ѝ картографиране. Навлизането на 2D лазерните скенери (далекомери) в мобилната роботика, предоставя на дизайнерите възможност за оптимални конструктивни решения, от гледна точка на това, че повечето съвременни скенери имат ъгъл на сканиране в диапазона 180 ÷ 270°, възможност за софтуерно ограничаване броя на сканиращите точки, с компактни размери са, а ориентацията на сензора влияе в допустимите граници върху точността на измерването,

позволяващо монтирането им да е на най-подходящото място в конструкцията на робота.

Голямата популярност на лазерните далекомери като средство за възприемане на работната среда се дължи на редица предимства пред останалите сензори за измерване на разстояния: осигуряват голям обем от данни при това с голяма точност на измерването; имат висока честота на дискретизация; малък дисперсионен ъгъл (0.5 ÷ 3.5mrad), и висока ъглова разрешаваща способност [5, 6].

В статията е представено практическо решение за реализация на *CDC хост* (*host*) за бордова комуникация с 2D лазерен скенер за визуализация на работната сцена на малък двуверижен мобилен робот с изисквания за минимизиране на енергийната консумация на всички бордови системи [7, 8, 9].

2D лазерен скенер

2D лазерният скенер *URG-04LX-UG01* (*Фиг.1*) [12], разполага само с *USB2.0* комуникационен интерфейс (*Таблица 1*), осигуряващ скорост на трансфер на данни до *12Mbps*.

Таблица 1. Спецификации на използваният сензор

Фиг.1.
Лазерен скенер

URG-04LX-UG01

Модел	**URG-04LX-UG01**
Обхват, [mm]	20 ÷ 4000
Точност	20÷1000mm: ±30mm; 20÷4000mm: ±3% от измерените стойности
Резолюция, [mm]	1
Ъгъл на сканиране, [°]	240
Ъглова резолюция, [°]	0.352
Ъглова скорост на сканиране, [°/sec]	360
Време за сканиране, [msec]	100
Източник на светлина	лазерен диод (λ=785nm)
Комуникационен интерфейс	USB2.0 (12Mbps)
Работни условия	-10~50°C влажност до 85%
Защита на корпуса по IP	Оптика - IP64; Тяло - IP40
Маса, [g]	160
Външни размери, mm	ш/д/в: 50/50/70
Захранване	USB
Мощност, [W]	2.5

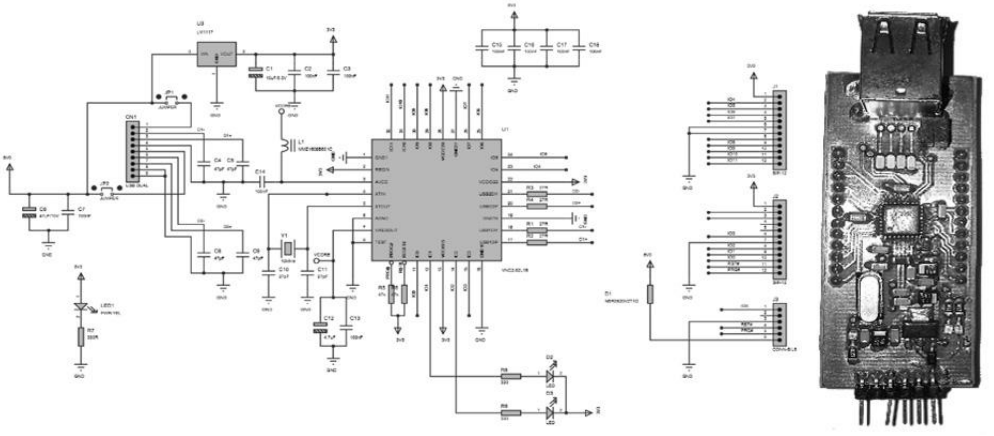
Скенерът се използва като основен източник за визуална информация, поради което надеждната комуникация със скенера е от първостепенно значение. Комуникационният протокол, имплементиран в сензора, се базира на *Communication Device Class* (*CDC*) *USB* протокола. Драйверите, необходими за комуникацията на скенера чрез този протокол, са вградени в *Windows OS* и когато скенерът е свързан директно към персонален компютър, операционната система сама инсталира необходимите драйвери и инициализира устройството, в следствие на което получаването на данни от него не представлява проблем.

Ограничението за минимална енергийна консумация на всичките бордови ресурси включително и изчислителните, налага да се изгради бордова микроконт-

ролерна система, поемаща функциите на персоналния компютър, без това да намалява производителността на далекомера и повишаване грешката на измерването от специфицираните (*Таблица 1*).

CDC хост контролер

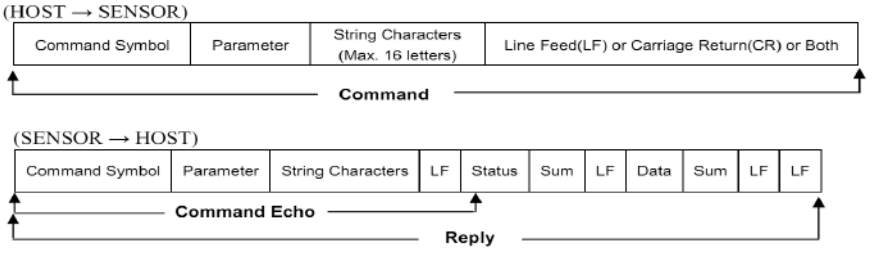
Принципната схема и практическата реализация на *CDC хост* контролер за управлението на сензора са показани на *Фиг. 2*. Модулът се използва като „посредник“ между бордовия контролер и лазерния скенер, като основното му предназначение е чрез софтуерна имплементация на *CDC* протокола да обезпечи необходимите инициализации и трансфер на команди към сензора и данни към оператора.



а - принципна схема *б - реализация*

Фиг. 2. Принципна схема и реализация на хост контролер

Реализацията използва микроконтролер *VNC2* на фирмата *FTDI* [14], който може да се използва като двоен *USB* хост и подчинено устройство, или две подчинени устройства. Когато се използва като *USB* хост, комуникацията му с други микроконтролери може да се осъществява посредством някой от другите имплементирани комуникационни интерфейси – *UART* или *SPI*. Бордовият контролер обменя информация с хоста посредством хардуерно имплементирания протокол *UART*, като в случая скоростта на трансфер е *256000Bps* (***B**aud **p**er **s**econd*), а формата на данните е *8N1* (*8* бита, *N* - без проверка по четност, *1* стоп-бит). Преди да се осъществи необходимата комуникация (бордови контролер – хост контролер) е необходимо *URG-04LX-UG01* да бъде разпознат като устройство, използващо *CDC* протокол и инициализирането му като устройство за данни. Веднъж инициализиран скенерът може да приема команди в съответствие с разработения от производителя протокол *SCIP2.0* (***S**canning sensor **C**ommand **I**nterface **P**rotocol*) версия 2 [13] и да връща в *ASCII* символи информация за разстоянията до измерените обекти в сцената. Общият формат на кадъра на протокола има вида показан на *Фиг. 3*. Инициирането на комуникацията става от управляващия бордови контролер. При разпознаване на съответната команда, скенерът отговаря, като показателно за правилно приетата команда е повторението на самата команда и след това подаването на съответните данни.



Фиг. 3. Комуникационен формат на SCIP2.0 протокола

URG-04LX-UG01 има възможност да разпознае *13* дефинирани в протокола команди (*Таблица 2*). Размерът на командите е *2Bytes*, а в полето параметър се указва конкретната информация, с която да се променят настройките на сензора или да се извлече допълнителна информация от него. Третото поле (*String Characters*) е опционално и обикновено в него се записва допълнителна информация от сензора, която да идентифицира например последователността на данните от множество сканирания. Полетата, обозначени с *LF* (*Line Feed*) или *CR* (Carriage Return), са терминиращи, като в командата може да се използват и двете, но сензора винаги завършва изпращането на пакета данни с два последователни *LF*. *Status* полето служи за идентифициране на работата на сензора. Полето е с размер *2Bytes* и кодове, различни от (*00*) или (*99*), са сигнал за грешка. Всеки ред с данни, изпратен от скенера, завършва с контролна сума – полето *Sum*, което е с размер *1Byte*, а в полето *Data* се записват фактическите данни от сензора, чийто размер е в зависимост от командата.

Таблица 2. Команди за управление на URG-04LX-UG01

Формат на команда	**Описание**
MD/(S)0044072500000\n	Команда за указване на сензора да започне сканирането, MD се използва за 3 байтово кодиране, а MS за двубайтово
G-(D)/(S)0044072500\n	Команда за указване на сензора да върне данните само от едно сканиране. Необходимо е преди това, лазера да се активира чрез командата BM
BM\n	Команда за включване на лазера на сензора
QT\n	Команда за изключване на лазера на сензора
RS\n	Команда за установяване на заводските настройки на сензора
TMx\n, x=0,1,2	Команда за разрешаване напасването на таймера на сензора (x=0), забраняването (x=2) или за получаване на текущата му стойност (x=1).
SS115200\n	Команда за установяване на скоростта на трансфер на данни когато сензора е свързан към сериен интерфейс
CRxx\n, xx=00÷10, 99	Команда за задаване скоростта на въртене на мотора, xx=00 за подразбиращата се скорост, а xx=99 указва да се възстанови заводски дефинираната скорост
HSx\n, x=0,1	Команда за превключване от нормален режим на чувствителността (x=0) на сензора в режим с висока чувствителност (x=1)
DBxx\n	Команда за симулиране на неизправност на сензора
VV\n	Команда за извличане на заводска информация за сензора, като сериен номер, версия на firmware и т.н.
PP/n	Команда указваща на сензора да изпрати спецификациите си сканиране
II/n	Команда указваща на сензора да изпрати текущите си настройки

Микроконтролерът *VNC2* работи с операционна система за реално време, а необходимата софтуерна реализация на *CDC хоста* е разгледана в следващата точка.

Софтуерна реализация на *USB CDC хост*

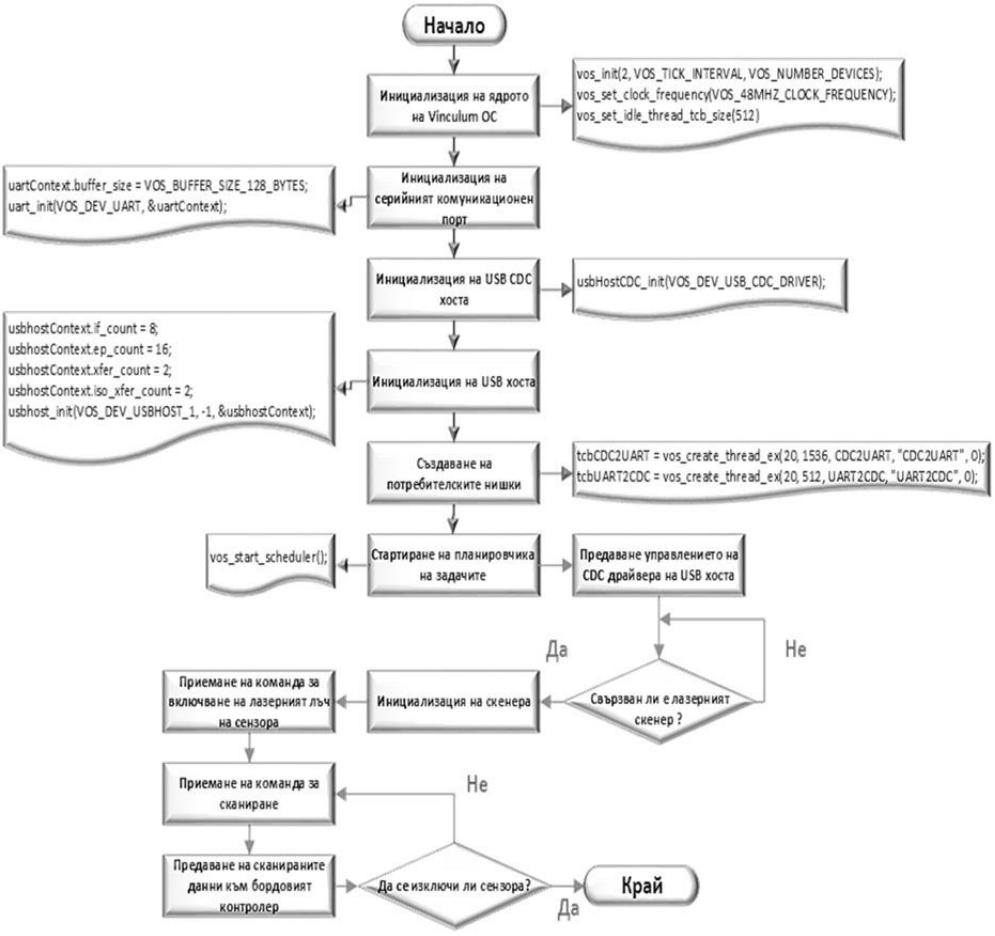
За софтуерната реализация на *CDC* хоста се използват вградените библиотечни функции на развойната среда *Vinculum II*. Йерархичната структура на реализацията е показана на *Фиг. 4*.

Както се вижда от фигурата, *CDC* драйверът се реализира като потребителско приложение и се разполага в слойния модел на *USB* хоста над *USB* хост драйвера. *VNC2* чипът работи с *VINCULUM RTOS* операционна система, която е приоритетна многозадачна операционна система. Управлението на отделните нишки става от *„планировчика - Kernel Scheduler*“ на задачите в ядрото на операционната система, който в зависимост от присвоения им приоритет, планира изпълнението им. Планировчикът също може да увеличава или намалява приоритета на потребителските нишки, да превключва между тях и да синхронизира работата им, когато информационният поток от една нишка служи като входна информация за друга.

Фиг. 4. Йерархична структура на реализираният драйвер

Преди да се използва *CDC* драйверът, той трябва да бъде инициализиран от операционната система, след което, контрола на информационния поток се осъществява от диспечера на задачите.

Софтуерната реализация на *CDC* хоста имплементира програмния алгоритъм показан на *Фиг. 5*.



Фиг. 5. Програмен алгоритъм на CDC хоста

В *Таблица 3* е пояснено предназначението на конкретните функции, потребителски нишки и манипулатори, използвани при реализацията на *CDC* хоста. Тъй като програмният код само за имплементацията на *CDC* хост драйвера надхвърля повече от *400* програмни реда, тук ще бъдат конкретизирани тези функции, които са от съществено значение за реализацията, а именно функциите ***CDC2UART()***, ***UART2CDC()*** и ***cdc_host_attach()***.

Когато главната функция ***main()*** извърши необходимите инициализации на ядрото, *USB* хост драйвера, *USB CDC* хост драйвера и входно-изходните портове на чипа, тя предава управлението на планировчика задачите на ядрото на VOS – *Kernel Scheduler*.

Таблица 3. Дефиниции на функции, нишки и манипулатори на CDC хоста

№	Декларация	Описание
1	`vos_tcb_t* tcbCDC2UART;` `vos_tcb_t* tcbUART2CDC;`	Указатели към потребителските нишки
2	`void CDC2UART();` `void UART2CDC();`	Функции за трансфер на данни от и към лазерният скенер

3	`VOS_HANDLE hUSBHOST_1;`
	`VOS_HANDLE hUART;`
	`VOS_HANDLE hUSB_CDC_DRIVER;`
		Манипулатори на операционната система VOS за USB хоста, серийният интерфейс и USB CDC драйвера

4	`void iomux_setup(void);`	Функция за инициализация на входно-изходните портове на VNC2 чипа

5	`void vos_init(uint8 quantum, uint16 tick_cnt, uint8 num_devices);`
	`void vos_set_clock_frequency(uint8 frequency);`
	`void vos_set_idle_thread_tcb_size(uint16 tcb_size);`
		Функции за инициализация ядрото на операционната система VOS и необходимата памет за потребителските нишки, които не са активни;

6	`unsigned char uart_init(unsigned char devNum, uart_context_t* context);`	Функция за инициализация на серийният интерфейс на VNC2 чипа
7	`unsigned char usbHostCDC_init(unsigned char devNum);`	Функция за инициализация на USB хост CDC драйвера и регистрирането му в диспечера на задачите
8	`unsigned char usbhost_init (unsigned char devNum_port_1, unsigned char devNum_port_2, usbhost_context_t* context);`	Функция за инициализация на USB хост драйвера и регистрирането му в диспечера на задачите.
9	`vos_tcb_t* vos_create_thread_ex(uint8 priority, uint16 stack, fnVoidPtr function, char* name, int16 arg_size);`	Функция за създаване на потребителските нишки, асоциирани с функциите за трансфер на данни от и към лазерният скенер
10	`void vos_init_mutex(vos_mutex_t* m,uint8 state);`	Функция за инициализация на примитив от тип „мутекс“ необходим за вътрешна синхронизация на VOS
11	`void vos_start_scheduler(void);`	Функция предаваща управлението към планировчика на задачите на ядрото на VOS

12	`unsigned char usbhost_connect_state(VOS_HANDLE hUSB);`
	`VOS_HANDLE cdc_host_attach(VOS_HANDLE hUSB, unsigned char devHostCDC);`
	`void cdc_host_detach(VOS_HANDLE hHostCDC);`
		Помощни функции за определяне свързаността на USB устройства, определяне класа на свързаното устройство и имплементираните комуникационни интерфейси, както и за освобождаване на заетите ресурси при освобождаване на устройството

13	`void open_drivers(void);`
	`void attach_drivers(void);`
	`void close_drivers(void);`
		Помощни функции за асоцииране на необходимите драйвери на VOS със съответните интерфейси и устройства

Основната задача на планировчика е да извиква регулярно през времевия интервал (***quantum***), указан във функция (5) потребителските функции (2). Първата функция ***void*** *CDC2UART()* е предназначена за осъществяване трансфера на данни от *CDC* устройството към серийния порт на контролера, чрез който хоста комуникира с бордовия микроконтролер. Според спецификацията за съответния клас устройства, максималният обем данни, който се предава за *1msec,* не може да надхвърля *64*Bytes. Това налага в паметта на контролера да се задели памет със съответния обем, необходима за съхранение на получените данни преди да бъдат изпратени през серийния интерфейс към бордовия контролер. Функцията има и грижата да извика помощната функция ***void*** *open_drivers(**void**)*, чрез която съответните драйвери за *USB* хоста и *UART-а* на *VNC2* се асоциират с манипулаторите на *VOS* операционната система. За да може да се осъществи необходимата комуникация, в тялото на функцията се извършват и необходимите настройки на UART, включващи задаването на:

Параметър	Програмна структура

• Бодова скорост	`uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_BAUD_RATE;`
	`uart_iocb.set.uart_baud_rate = UART_BAUD_115200;`
	`vos_dev_ioctl(hUART, &uart_iocb);`

• Контрол на трансфера	`uart_iocb.ioctl_code =`
	`VOS_IOCTL_UART_SET_FLOW_CONTROL;`
	`uart_iocb.set.param = UART_FLOW_NONE;`
	`vos_dev_ioctl(hUART, &uart_iocb);`

• Брой битове данни	`uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_DATA_BITS;`
	`uart_iocb.set.param = UART_DATA_BITS_8;`
	`vos_dev_ioctl(hUART, &uart_iocb);`

• Брой на стоп битовете	`uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_STOP_BITS;`
	`uart_iocb.set.param = UART_STOP_BITS_1;`
	`vos_dev_ioctl(hUART, &uart_iocb);`

• Контрол по четност	`uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_PARITY;`
	`uart_iocb.set.param = UART_PARITY_NONE;`
	`vos_dev_ioctl(hUART, &uart_iocb);`

След инициализацията на *UART*, потребителската нишка изчаква да се осъществи разпознаването (*enumeration*) на съответния интерфейс, клас, подклас и прилежащия им протокол, имплементирани при производството на скенера, обслужващи лазерния скенер, като за целта манипулаторът ***hUSB_CDC_DRIVER*** (3) се инициализира посредством функция ***cdc_host_attach()*** (12). При разпознаване на скенера като *CDC* устройство, нишката извлича информация за кодирането на линията и активното ниво на сигнала (логическа *0* или *1*) и при липса на грешка сигнализира на планировчика на задачите, че инициализацията е преминала и другите потребителски нишки могат да заемат мястото ѝ. Ако потребителската нишка не бъде деактивирана, се проверява дали има данни, които да се трансферират и с какъв обем са. При наличие на актуални данни те се записват във временния буфер, след което се изпращат през серийния интерфейс на *VNC2* чипа към бордовия контролер. Индикация за работата на нишката се получава чрез управлението на два светодиода, свързани към портовете на чипа (*Фиг.2.а*). В следващия програмен код, процедурата за индикация е описана само веднъж, а редовете, в които отново се повтаря, е заместена с многоточие.

```
/*************************************************************************
/
       do{
       //установи какво е количеството данни от лазерният скенер
       CDC_iocb.ioctl_code = VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS;
       status = vos_dev_ioctl(hUSB_CDC_DRIVER, &CDC_iocb);
       if (status != USBHOSTCDC_OK){
              leds = LED5;
       vos_gpio_write_port(GPIO_PORT_A, leds);
       break;
       }
       dataAvail = CDC_iocb.get.queue_stat;
       if (dataAvail > 0){
       //Прочети колко байта реално са записани във временният буфер
       status    =    vos_dev_read(hUSB_CDC_DRIVER,    buffer,    dataAvail,
&actual);
       if (status != USBHOSTCDC_OK){...}
       //Изпрати записаните данни през UART-а на VNC2
       if (actual > 0){
       leds = 0;
       vos_gpio_write_port(GPIO_PORT_A, leds);
       status = vos_dev_write(hUART, buffer, actual, NULL);
       if (status != UART_OK){...}
       leds = LED4;x
       vos_gpio_write_port(GPIO_PORT_A, leds);
       }
       }
       }while (1);
/*************************************************************************
/
```

Процедурата се повтаря докато диспечера не я блокира или времето, през което може да е активна, не измине.

Другата потребителска нишка ***UART2CDC()*** (2) от *Таблица 3*

Таблица има аналогично действие в частта на трансфериране на данните. За разлика от ***CDC2UART()*** нишката, тази нишка първоначално изчаква сигнал за приключване на инициализацията на другите нишки, след което прави запитване дали има постъпили валидни данни в приемния буфер на *UART-a*. Ако, по някаква причина, постъпилите данни са в размер по-голям от размера на буфера на *CDC* нишката, данните, надвишаващи го, биват премахнати. Останалите валидни данни се прочитат и се изпращат към *CDC* хоста, а края на трансфера се инициира с индексиране на мутекс примитива.

Следващият програмен код демонстрира програмната реализация на тази функция, като тук отново кодът, даващ индикация за грешка, е заместен с многоточие:

```
/*************************************************************************
/
       void UART2CDC(){
       unsigned char status;                //променлива за индикация на
грешки
       unsigned char buffer[64];            //буфер за постъпващите данни
       common_ioctl_cb_t uart_iocb;         // достъп до UART структурата
       unsigned short dataAvail, actual;    //достъпни и реално прието
данни
       unsigned char leds;
```

```
        do{
        // изчакай другите нишки да се приключат с инициализацията си
        vos_lock_mutex(&mInit);
        uart_iocb.ioctl_code = VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS;
        vos_dev_ioctl(hUART, &uart_iocb);
        dataAvail = uart_iocb.get.queue_stat;
        if (dataAvail > sizeof(buffer)){dataAvail = sizeof(buffer);}
        if (dataAvail > 0){
        // прочети данните от UART-а
        status = vos_dev_read(hUART, buffer, dataAvail, &actual);
        if (status != UART_OK){...}
        leds = 0;
        vos_gpio_write_port(GPIO_PORT_A, leds);
        // предай получените данни към CDC хоста
        status = vos_dev_write(hUSB_CDC_DRIVER, buffer, actual, NULL);
        if (status != USBHOSTCDC_OK){...}
        leds = LED4;
        vos_gpio_write_port(GPIO_PORT_A, leds);
        }vos_unlock_mutex(&mInit);
        }while (1);}
/**************************************************************************
/
```

Трета функция със значение при реализацията на USB CDC хоста е ***VOS_HANDLE*** *cdc_host_attach(**VOS_HANDLE** hUSB, **unsigned char** devHostCDC)*. Чрез нея става инициализацията на лазерния скенер, като функцията връща след изпълнението си манипулатор, асоцииран с *VOS* операционната система. Основната ѝ роля е да осигури всички необходими дескриптори на *CDC* класа и да направи проверка дали включеното устройство съответства на дефинираните параметри за класа. Функцията има следната програмна последователност в съответствие със стандарта за *USB CDC* комуникационните устройства:

Действие	Програмна структура

Дефиниране на класа на устройството	`hc_iocb_class.dev_class = USB_CLASS_CDC_CONTROL;`
	`hc_iocb_class.dev_subclass=`
	`USB_SUBCLASS_CDC_CONTROL_ABSTRACT;`
	`hc_iocb_class.dev_protocol = 1;`

Намиране на първият интерфейс от потребителската нишка	`hc_iocb.ioctl_code =`
	`VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;`
	`hc_iocb.handle.dif = NULL;`
	`hc_iocb.set = &hc_iocb_class;`
	`hc_iocb.get = &ifCDCControl;`

Намиране на интерфейса за данни на скенера	`hc_iocb.ioctl_code =`
	`VOS_IOCTL_USBHOST_DEVICE_GET_NEXT_HANDLE;`
	`hc_iocb.handle.dif = ifCDCControl;`
	`hc_iocb.get = &ifCDCData;`

Проверка дали скенера в действителност е устройство за данни	`hc_iocb.ioctl_code =`
	`VOS_IOCTL_USBHOST_DEVICE_GET_CLASS_INFO;`
	`hc_iocb.handle.dif = ifCDCData;`
	`hc_iocb.get = &hc_iocb_class;`

Проверка за съответствие между необходимият USB клас и класа на сензора	`if ((hc_iocb_class.dev_class !=`
	`USB_CLASS_CDC_DATA) ||`
	`(hc_iocb_class.dev_subclass != 0) ||`
	`(hc_iocb_class.dev_protocol != 0)){return NULL;}`

След като премине по-горе описаните процедури, функцията инициализира *CDC* драйвера със сензора, като предава контрола на *USB* хоста. Ако сензора не отговаря на съответните изисквания, функцията връща нулев указател към потребителската нишка, в следствие на което *USB* хостът приема, че включеното устройство не отговаря на изискванията и не стартира процедурата за извличане на състоянието на *CDC* комуникационната линия.

Създаването на потребителските нишки става чрез функция (9) от *Таблица 3*, като в случая имат следната дефиниция:

```
tcbCDC2UART = vos_create_thread_ex(20, 1536, CDC2UART, "CDC2UART", 0);
tcbUART2CDC = vos_create_thread_ex(20, 512, UART2CDC, "UART2CDC", 0);.
```

За да са равнопоставени една на друга, двете нишки имат приоритет *20*, като основната разлика се състои в паметта, необходима за имплементирането им. Това е така, защото в тялото на функцията *CDC2UART()* е поместен и кодът за инициализация на *UART*-а на чипа и извличане на информацията за състоянието на линията. Третият параметър е указател към функцията, с която да се асоциира съответната нишка, а четвъртият, с който се задава името на нишката, служи основно за обозначението ѝ в режим на настройка на програмния код в програмната среда.

На Фиг. 6 е показан *screenshot* от съвместната работа на отделните нишки управлявани от имплементираната в чипа *RTOS –VOS.*

Thread Manager

#	Thread	Priority	State	Thread Type	CPU (%)	Peak Stack (Bytes)	Current Stack (Bytes)	Quantum	Thread Address
0	VOS Idle Thread	0	Ready	Idle Thread	0.59		8	2	0x1af
1	HCR	∞	Blocked	System Thread	0.00		17	2	0x837
2	HCD	∞	Blocked	System Thread	0.04		17	2	0xa3b
3	HCH	∞	Delayed	System Thread	3.99		16	2	0xb3f
4	CDC2UART	20	Ready	Application Thread	47.67		206	0	0xd43
5	UART2CDC	20	Running	Application Thread	47.68		110	1	0x1347
6	UHCDCSP	31	Blocked	Application Thread	0.00		125	2	0x158f
7	UHCDCP	30	Blocked	Application Thread	0.00		165	2	0x18d7

Фиг. 6. Мениджър на нишките управлявани от VINCULUM RTOS

На ред *№5* е показана нишката, която се изпълнява в момента от операционната система, в редове *№0* и *№4* са нишките, които са готови да се стартират. На ред *№3* е маркирана забавената нишката, а останалите нишки са блокирани. Както стана ясно, за управлението на нишките, *VOS* използва два параметъра:

- **uint8** quantum – дефинира времето за превключване между отделните нишки в *msec*;
- **uint16** tick_cnt – параметър, задаващ стойността на инкрементиране на таймерният модул на чипа в *msec*;

По подразбиране параметърът „quantum“ е настроен за *50ms*, което означава, че всяка нишка ще е активна през такъв интервал от време. В случая този параметър не удовлетворява изискванията за бърз обмен на данни между бордовия контролер и скенера, в следствие на което превключването на съответните нишки, както се вижда от *Фиг. 6,* е променено да става през интервал от *2ms*. Посочената промяна напълно удовлетворява изискванията за предаването на данните в реално време.

Заключение

В статията са описани специфичните особености при изграждането на бордови CDC хост контролер за комуникация с 2D лазерен скенер с приложение за мобилни роботи и в частност – малък двуверижен робот. Основните изисквания за минимална

енергийна консумация от една страна и ограничените възможности за комуникация с 2D лазерния скенер, както и малкия наличен монтажен обем на робота, са основният мотив да се прибегне до изграждането на посочената микроконтролерна система. Подробно са описани специфичните комуникационни изисквания и софтуерната имплементация на USB CDC хоста, а проведените експерименти за съвместяването на работата на проектирания хардуер и софтуер показват, че така проектираната система може да осигури предаване на данни в реално време, без това да доведе до увеличаване на грешката на измерването сравнена с реферираната от производителя на използвания сензор.

Литература:

1. Rahnavard, M. et al., ***„RoboCupRescue 2006 - Robot League Team Ariana (Iran)“*** Tech. rep. , p. 97 2006.
2. Styrstrom, D. and Holmqvist, M., ***„RoboCupRescue 2006 - Robot League Team RFC Uppsala (Sweden)“*** Tech. rep. , p. 11 2006.
3. Birk, A., Pathak, K., and Schwertfeger, S., ***„The IUB Rugbot: an intelligent, rugged mobile robot for search and rescue operations“*** in *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR)*, 2006, p. 6. [Online]. http://robotics.iu-bremen.de/publications/SSRR2006_Rugbot.pdf
4. Abbott, L. et al., ***„RoboCupRescue 2006 - Robot League Team Good Samaritan Urban Search and Rescue Robot (USA)“*** *Mechanical Engineering*, 2006.
5. Kneip, L., Tache, F., Caprari, G., and Siegwart, R., ***„Characterization of the compact Hokuyo URG-04LX 2D laser range Scanner“*** *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1447-1454, 2009.
6. Okubo, Y., Ye, C., and Borenstein, J., ***„Characterization of the Hokuyo URG-04LX laser rangefinder for mobile robot obstacle negotiation“****Proceedings of SPIE*, vol. 7332, pp. 733212:1--733212:10, 2009. [Online]. ://WOS:000292646300026
7. Заманов, Вл, Димитров, Ат., Симеонов, Ст., ***„Развитие на сензорната система на изследователски мобилен робот“*** *Сборник с доклади от международна конференция Автоматика'2012*, годишник на технически университет - София, том 62, книга 2, 2012, с.303-312, ISSN 1311-0829
8. Заманов, Вл., Димитров, Ат. ***„3D сканиране с мобилент робот и 2D лазарен далекомер“****, Механика на Машините,* год. XXII, книга 2, 2014, с. 26-29, ISSN 0861-9727
9. Dimitrov, At., Zamanov, V., ***„Ranging system of remote control tracked robot, Robotics****, Automation and Mechatronics'12,* Sofia, 15-17 October, 2012, pp. r1-r4, 2012, ISSN 1314-463
10. Maxbotix Inc. MB1020 LV-MaxSonar®-EZ2™ High Performance Ultrasonic Rangefinder. http://www.maxbotix.com/Ultrasonic_Sensors/MB1020.htm
11. Sharp Corporation, „GP2Y0D805Z0F Distance Measuring Sensor Unit Digital output (50 mm) type“ , 2006, Datasheet
12. URG-04LX-UG01 Product specification https://www.hokuyo-aut.jp/dl/URG-04LX_UG01_Specification.pdf
13. Kawata, H., Ohya, A., Yuta , , Santosh, W., and Mori, T., ***„Development of ultra-small lightweight optical range sensor system"*** in *Intelligent Robots and Systems*, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on, 2005, pp. 1078-1083.
14. Datasheet Vinculum-II Embedded Dual USB Host Controller IC Version 1.7, http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_Vinculum-II.pdf