LABORATION

Datorteknik Y

Kodlås på Darma

Version 3.5
January 2025 (AN, KP)

Namn och personnummer	**Godkänd**

1 Inledning

Syftet med laborationen är att ge övning i assemblerprogrammering samt att skapa nödvändig kännedom om laborationsutrustningen inför senare laborationer i denna kurs. Efter den här laborationen kommer du att ha bra koll på instruktionsuppsättningen på ARM samt hur du hanterar in och utmatning på TiVA C LaunchPad som vi använder. Du ska även vara förtrogen med labsystemet programmeringsmiljön Code Composer Studio och dess kommandon.

Innan du tar itu med förberedelserna bör du åtminstone läsa kapitel 2, 3 och avsnitt 4.1, 4.2 och 4.4 i häftet *Laborationsmiljön Darma*.

Till laborationen ska du ha med dig lösningsförslag på samtliga obligatoriska uppgifter i detta labhäfte samt åtminstone två av de extrauppgifter som ingår. (Det är obligatoriskt att på laborationen genomföra åtminstone en av dessa extrauppgifter.)

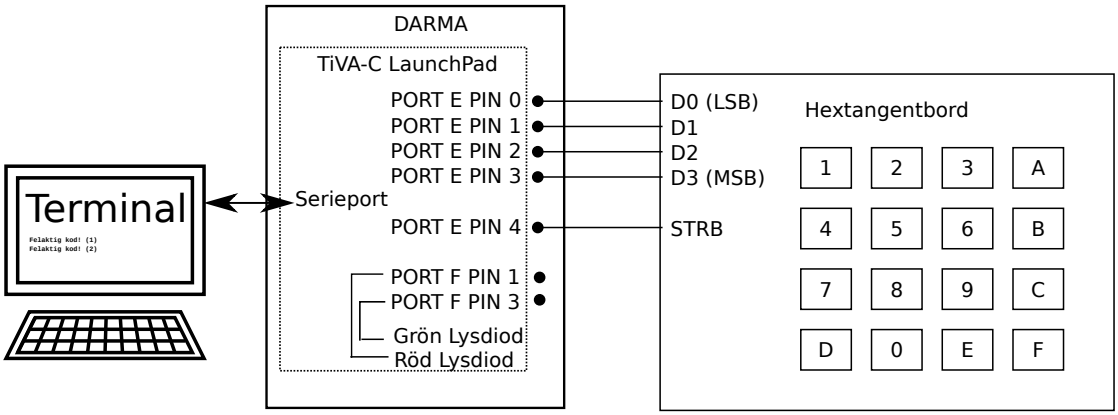
2 Specifikation av kodlåsets funktion

I denna laboration ska du implementera ett enkelt kodlås i assembler på labsystemet. Kodlåset är tänkt att sitta i ett större system där det används för att slå på respektive av ett larm av något slag. Att larmet är aktiverat visar vi i denna laboration genom att en lysdiod lyser rött. Om larmet inte är aktivt ska lysdioden istället lysa grönt. Ett hextangentbord med knapparna 0–9 och A–F används för att aktivera respektive avaktivera larmet enligt följande specifikation:

- Larmet start i aktivt läge (dvs låst)
- När larmet är aktivt ska du kunna trycka in minst en fyrsiffrig kod (decimala siffror) följd av tangenten **F** för att avaktivera larmet. Om rätt kod har skrivits in så ska lysdioden som indikerar att larmet är på byta färg till grönt. Om felaktig kod skrivs in så ska istället texten *"Felaktig kod!"* skrivas ut på terminalen.
- Det är de senaste fyra siffrorna som räknas. Det vill säga, om rätt kod är "6510" så ska man exempelvis kunna trycka in sekvensen "6526510F" för att låsa upp dörren.
- När låset är upplåst ska det bara reagera om du trycker på tangenten **A** på knappsatsen, vilket ska aktivera larmet på kodlåset (dvs lysdioden ska då lysa rött).

3 Inkoppling av knappsats och lysdiod

LaunchPad-kortet (det röda) har en flerfärgs lysdiod som ska användas för att visa om larmet är aktivt (röd färg) respektive inaktivt (grön färg). Styrsignalerna till denna lysdiod är bit 1 och 3 av port F, men eftersom lysdioden är monterad på LaunchPad-kortet behöver ingen koppling göras. Däremot visar lysdioderna för bit 1 och 3 på Port F vilken färg som ska lysa.



Figur 1: Labuppkopplingen

Knappsatsen (det hexadecimala tangentbordet) ska kopplas in enligt figur 1. Knappsatsen har fem signaler som ska kopplas in (utöver jord och matningsspänning). Fyra av dessa signaler (D0-D3) visar vilken knapp som tryckts ner på hextangentbordet. Den sista signalen (STRB) signalerar att en tangent trycks ner genom att gå hög. Denna signal är avstudsad.

För att förbereda laborationen utan tillgång till labblokalen (och vid distansläge) kan istället en speciellt ansluten hårdvara användas. Denna består av ett TiVA C LaunchPadkort (det röda kortet) tillsammas med en Arduino Uno som styrs via programmet tsea28lab1. Programmet tsea28lab1 styr då Arduino Uno som skickar knapptryck och visar färgen på LED.

4 Minnesadresser att använda

De senaste fyra siffrorna som skrivits in på hextangentbordet ska sparas i minnet. Den förväntade sekvensen av hextangenttryck ska också lagras i minnet. Lagra varje knapptryck i var sin byte. Lagra även textsträngen Felaktig kod!"i programminnet.

Om du använder de här minnesadresserna så blir det lättare för labassistenterna att hjälpa dig om du får problem:

- någonstans i programminnet: Här ska textsträngen *Felaktig kod!* finnas lagrad med hjälp av ASCII-kodning, följd av koden för newline och carriage return. Använd en label för att kunna referera till texten (se avsnitt 4.4 i [1])
- 0x20001000–0x20001003: Här ska du lagra de fyra senaste siffrorna som skrivits in på hextangentbordet. (Den tangent som tryckts in senast lagras på adress 0x20001000.)
- 0x20001010–0x20001013: Här ska den korrekta koden finnas lagrad. Första siffran i koden placeras i 0x20001013 och sista siffran i 0x20001010.

Den rätta koden behöver sättas av programmet. Detta ska göras i början av programmet.

5 Subrutiner

Det här avsnittet innehåller information om de subrutiner som vi vill att du minst delar upp ditt program i samt befintliga subrutiner. Förutom huvudprogrammet så ska du skriva åtminstone 7 subrutiner själv.

5.1 Givna subrutiner

- `inituart`: Initiera serieport
- `initGPIOE`: Initiera GPIO port E
- `initGPIOF`: Initiera GPIO port F
- `printchar`: Skriv ut ett tecken på terminalen

Dessa subrutiner finns med i labbmaterialet du laddar ned från hemsidan. Nedan följer en beskrivning av vad varje rutin gör:

5.2 printchar - Utskriftsrutin

Följande subrutin skriver ut ett ASCII-kodat tecken i register r0 på terminalen.

```
;; Utskrift av ett tecken på serieport
;; r0 innehåller tecken att skriva ut (1 byte)
;; returnerar först när tecken skickats
;; förstör r1 och r2
printchar:
    mov r1,#(UART0_UARTFR & 0xFFFF)  ; peka på serieportens
    movt r1,#(UART0_UARTFR >> 16)     ; statusregister
loop1:
    ldr r2,[r1]          ; hämta statusflaggor, bit 5 =1 om upptagen
    ands r2,r2,#0x20     ; kan ytterligare tecken skickas (bit5=0)?
    bne loop1            ;   nej, försök igen
    mov r1,#(UART0_UARTDR & 0xFFFF)  ; peka på serieportens
    movt r1,#(UART0_UARTDR >> 16)     ; dataregister
    str r0,[r1]          ; skicka tecken
    bx  lr
```

5.3 inituart - Initiera serieport

Denna subrutin initierar serieporten så tecken kan skickas till datorn. Denna behöver anropas en gång vid starten av programmet.

5.4 initGPIOE - Initiera GPIO port E

Denna subrutin ställer in GPIO port E till ingångar. Denna rutin måste anropas 1 gång vid starten av programmet.

5.5 initGPIOF - Initiera GPIO port F

Denna subrutin ställer in GPIO port F. Bit 0 och 4 sätts till ingångar, och bit 1-3 sätts in till utgångar. Denna rutin måste anropas 1 gång vid starten av programmet.

5.6 Subrutiner att skriva

Följande lista av subrutiner ska implementeras enligt beskrivningen nedan. Dvs skapa subrutiner som gör vad som anges i beskrivningen nedan, men inte mer än vad som beskrivs.

- `printstring`: Skriv ut en textsträng
- `deactivatealarm`: Avaktivera larmet
- `activatealarm`: Aktivera larmet
- `getkey`: Hämta tecken från hextangentbordet
- `addkey`: Lägg till tangent i inbuffern
- `clearinput`: Rensa inbuffer
- `checkcode`: Kolla om rätt kod tryckts in

När du kommer till laborationen ska du ha förberett assemblerkod för alla dessa subrutiner. *Du ska givetvis förbereda huvudprogrammet i förväg också.* Du får gärna lägga till fler subrutiner om du känner att du behöver det.

5.7 printstring - Skriv ut en textsträng

Den här subrutinen ska skriva ut en textsträng på terminalen (med hjälp av subrutinen `printchar`). Subrutinen ska kunna skriva ut vilken strängs som helst som r4 pekar på. Antal tecken i strängen ska ha angetts i r5.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Inargument: Pekare till strängen i r4
;             Längd på strängen i r5
; Utargument: Inga
;
; Funktion: Skriver ut strängen mha subrutinen printchar
printstring:
     ; Förberedelseuppgift: Skriv denna subrutin!
     bx  lr
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

5.8 deactivatealarm - Avaktivera larm

Den här subrutinen ska få lysdioden lysa grönt.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Inargument: Inga
; Utargument: Inga
;
; Funktion: Tänder grön lysdiod (bit 3 = 1, bit 2 = 0, bit 1 = 0)
deactivatealarm:
    ; Förberedelseuppgift: Skriv denna subrutin!
    ...
    ...
    bx  lr
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

5.9 activatealarm - Aktivera larm

Den här subrutinen ska få lysdioden att lysa rött.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Inargument: Inga
; Utargument: Inga
;
; Funktion: Tänder röd lysdiod (bit 3 = 0, bit 2 = 0, bit 1 = 1)
activatealarm:
    ; Förberedelseuppgift: Skriv denna subrutin!
    ...
    ...
    bx  lr
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

5.10 getkey - Hämta tecken från hextangentbordet

Den här subrutinen ska vänta på att användaren trycker på en tangent på hextangentbordet. Bara strobe-signalen ifrån tangentbordet ska användas för att avgöra om en tangent trycks ned (bit 7-5 ska inte kunna påverka funktionen). I del A får denna subrutin inte anropa andra subrutiner. Tips: Se till att du returnerar ifrån denna subrutin först när användaren slutar trycka på tangenten!

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Inargument: Inga
; Utargument: Tryckt knappt returneras i r4
getkey:
    ; Förberedelseuppgift: Skriv denna subrutin!
     ...
     ...
     bx  lr
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

5.11 addkey - Lägg till tangent i inbuffern

Den här subrutinen ska lägga till ett tecken i inbuffern genom att skifta buffern framåt ett steg. (Se exemplet nedan.)

Ursprungstillstånd		Efter det att addkey anropats med värdet 1 i r4	
0x20001000:	0x05	0x20001000	0x01
0x20001001:	0x06	0x20001001	0x05
0x20001002:	0xFF	0x20001002	0x06
0x20001003:	0xFF	0x20001003	0xFF

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Inargument: Vald tangent i r4
; Utargument: Inga
;
; Funktion: Flyttar innehållet på 0x20001000-0x20001002 framåt en byte
; till 0x20001001-0x20001003. Lagrar sedan innehållet i r4 på
; adress 0x20001000.
addkey:
    ; Förberedelseuppgift: Skriv denna subrutin!
    ...
    ...
    bx  lr
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

5.12 clearinput - Rensa inbuffer

Den här subrutinen är tänkt att lägga in en ogiltig kod i inbuffern.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Inargument: Inga
; Utargument: Inga
;
; Funktion: Sätter innehållet på 0x20001000-0x20001003 till 0xFF
clearinput:
    ; Förberedelseuppgift: Skriv denna subrutin!
    ...
    ...
    bx  lr
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

5.13 checkcode - Kolla om rätt kod tryckts in

Den här subrutinen kollar om den kod som ligger i inbuffern på 0x20001000 är korrekt. Denna subrutin ska bara göra jämförelsen, inte anropa andra subrutiner

eller läsa port F. *Krav: Du får enbart göra en läsning ifrån 0x20001000–0x20001003 respektive 0x20001010–0x20001013 i denna subrutin.*

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Inargument: Inga
; Utargument: Returnerar 1 i r4 om koden var korrekt, annars 0 i r4
checkcode:
    ; Förberedelseuppgift: Skriv denna subrutin!
    ...
    ...
    bx  lr
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

6 DEL 0: Testa subrutinerna var för sig

Innan du försöker implementera hela kodlåset är det smart att testa både de givna subrutinerna och dina egna subrutiner var för sig. Här nedan finns tips på hur du skriver olika versioner av main som testar de subrutiner som vi rekommenderar att du använder i denna laboration.

6.1 printchar

- Anropa inituart
- Definiera label endloop
- Sätt `r0` till 64
- Anropa printchar
- Avsluta programmet med ett återhopp till nästa utskrift enligt följande: `b endloop`
- När du kör programmet ska en massa snabela (@) skrivas ut.

Ett test av utskrift av enskilt tecken kan också vara lämpligt:

- Anropa inituart
- Sätt `r0` till 65
- Anropa printchar
- Avsluta programmet med en oändlig loop enligt följande kodsnutt: `endloop: b endloop`
- När du kör programmet ska ett stort A skrivas ut.

6.2 initGPIOE

- Anropa initGPIOE
- Avsluta programmet med en oändlig loop (enligt beskrivning ovan)
- Kör programmet.
- I fönstret "Registers" tittar du på registret GPIO_PORTE– GPIO_DATA. Tryck på en tangent på tangentbordet. Tryck på refresh-knappen (gul ikon till höger). Det ska då gå att se vilken tangent som tryckts.

6.3 initGPIOF

- Anropa initGPIOF
- Avsluta programmet med en oändlig loop (enligt beskrivning ovan)
- Kör programmet.
- I fönstret "Registers" tittar du på registret GPIO_PORTF– GPIO_DATA. Ändra värdet i GPIO_DATA till 0x02. Den röda lysdioden ska då tändas. Testa även med värdet 0x08.

6.4 printstring

- Anropa inituart
- Sätt r4 till 0x010000c0
- Sätt r5 till 13
- Anropa printstring
- Avsluta programmet (enligt beskrivning ovan)
- När du kör programmet ska strängen *Copyright (C)* skrivas ut.

6.5 deactivatealarm och activatealarm

- Anropa initGPIOF
- Anropa activatealarm
- Avsluta programmet enligt ovan.
- När du kör programmet nu så ska lysdioden lysa rött.
- Ändra programmet så att du anropar deactivaetalarm istället för activatealarm.
- När du kör programmet nu så ska lysdioden lysa grönt.

6.6 getkey

- Anropa initGPIOE
- Anropa getkey
- Avsluta programmet enligt ovan
- Kör programmet, tryck på valfri knapp. Tryck därefter på paus. Nu ska du i registerdumpen kunna se vilken knapp du tryckte in genom att titta på innehållet i r4. Notera att programmet inte ska avslutas förräns du *släppt* tangenten.
- Anslut en etta på bit 5 på port E, provkör programmet igen. Det ska fortfarande fungera som innan.

6.7 clearinput

- Anropa clearinput
- Avsluta programmet enligt ovan
- Verifiera att 0x20001000–0x20001003 innehåller värdet `FF FF FF FF`.

6.8 addkey

- Anropa clearinput
- Sätt r4 till 1
- Anropa addkey
- Sätt r4 till 2
- Anropa addkey
- Sätt r4 till 3
- Anropa addkey
- Sätt r4 till 4
- Anropa addkey
- Avsluta programmet enligt ovan
- Undersök om innehållet i 0x20001000–0x20001003. Det ska innehålla `04 03 02 01`

6.9 checkcode

- Anropa checkcode
- Avsluta programmet enligt ovan
- Innan du kör igång detta program så använder du memory browser för att skriva in två koder som är identiska på position 0x20001000–0x20001003 respektive 0x20001010–0x20001013. När du kör programmet och sedan tryck på paus så ska du i registerdumpen se att r4 fick värdet 1.
- Ändra i minnet så att koderna inte är identiska. Kör sedan programmet igen och verifiera att r4 innehåller 0.

6.10 Trace och brytpunkter

Tänk också på att du kan använda step into, step over samt sätta brytpunkter för att stega dig igenom programmet.

7 DEL A: Kodlåset

Nu när du har verifierat att alla dina subrutiner fungerar kan du kombinera dessa i ett huvudprogram som implementerar kodlåset. Tänkt på att ej anslutna pinnar/bitar i portarna kan få olika värden mellan varje läsning. Port F ska inte läsas i programmet, bara skrivas till mha subrutinerna.

8 DEL B: Varianter på kodlåset

Inför laborationen så ska du förbereda minst två av dessa varianter.[1] Du behöver dock enbart redovisa en variant på laborationstillfället. *Tips: Skapa ett nytt projekt för varje version- Om du lägger till eller modifierar subrutiner, testa dessa subrutiner separat på samma sätt som du testade dina övriga subrutiner innan du lade in dessa i huvudprogrammet!*

8.1 Blinkande lysdiod

Istället för att lysdioden lyser med fast sken när larmet är aktiverat så ska lysdioden blinka med en ungefärlig frekvens på 1 Hz. Processorn utför ca 16 miljoner instruktion per sekund.

8.2 Tidsbegränsad öppning

När du avaktiverat larmet så ska larmet aktiveras igen efter cirka fem sekunder. Så snart en knapp (förutom **A**) trycks ner så nollställs denna tidsfördröjning. Trycks A ned så ska låset aktiveras direkt.

[1] Detta för att du lätt ska kunna byta till en annan variant på laborationen om det visar sig att koden för den första varianten du provat är helt fel.

8.3 Möjlighet att byta kod

Det ska vara möjligt att byta kod på låset genom att trycka in den nya koden två gånger i rad och sedan trycka på tangenten C på hextangentbordet. Om samma kod trycktes in två gånger i rad så ska larmet aktiveras och koden bytas. Om inte samma kod trycktes så ska ingenting hända. Byte av kod ska enbart vara möjligt att göra när larmet är avaktiverat. Om något annat än 8 decimala siffror matas in före C ska larmet aktiveras.

8.4 Bättre felmeddelanden till terminalen

När textsträngen *"Felaktig kod!"* skrivs ut på terminalen så ska du också skriva ut antalet gånger som felaktid kod skrivits in. (Räknaren nollställs när rätt kod skrivs in.) Du behöver kunna hålla ordning på max 63 felaktiga försök. (Men det gör inget om du håller ordning på fler.) Antalet felaktiga försök ska skrivas ut decimalt!

9 ASCII-tabell

0x00	NUL '\0'	0x20	SPACE	0x40	@	0x60	‘
0x01	SOH (start of heading)	0x21	!	0x41	A	0x61	a
0x02	STX (start of text)	0x22	”	0x42	B	0x62	b
0x03	ETX (end of text)	0x23	#	0x43	C	0x63	c
0x04	EOT (end of transmission)	0x24	$	0x44	D	0x64	d
0x05	ENQ (enquiry)	0x25	%	0x45	E	0x65	e
0x06	ACK (acknowledge)	0x26	&	0x46	F	0x66	f
0x07	BEL '\a' (bell)	0x27	´	0x47	G	0x67	g
0x08	BS '\b' (backspace)	0x28	(	0x48	H	0x68	h
0x09	HT '\t' (horizontal tab)	0x29	)	0x49	I	0x69	i
0x0A	LF '\n' (line feed)	0x2A	*	0x4A	J	0x6A	j
0x0B	VT '\v' (vertical tab)	0x2B	+	0x4B	K	0x6B	k
0x0C	FF '\f' (form feed)	0x2C	,	0x4C	L	0x6C	l
0x0D	CR '\r' (carriage ret)	0x2D	-	0x4D	M	0x6D	m
0x0E	SO (shift out)	0x2E	.	0x4E	N	0x6E	n
0x0F	SI (shift in)	0x2F	/	0x4F	O	0x6F	o
0x10	DLE (data link escape)	0x30	0	0x50	P	0x70	p
0x11	DC1 (device control 1)	0x31	1	0x51	Q	0x71	q
0x12	DC2 (device control 2)	0x32	2	0x52	R	0x72	r
0x13	DC3 (device control 3)	0x33	3	0x53	S	0x73	s
0x14	DC4 (device control 4)	0x34	4	0x54	T	0x74	t
0x15	NAK (negative ack.)	0x35	5	0x55	U	0x75	u
0x16	SYN (synchronous idle)	0x36	6	0x56	V	0x76	v
0x17	ETB (end of trans. blk)	0x37	7	0x57	W	0x77	w
0x18	CAN (cancel)	0x38	8	0x58	X	0x78	x
0x19	EM (end of medium)	0x39	9	0x59	Y	0x79	y
0x1A	SUB (substitute)	0x3A	:	0x5A	Z	0x7A	z
0x1B	ESC (escape)	0x3B	;	0x5B	[	0x7B	{
0x1C	FS (file separator)	0x3C	<	0x5C		0x7C	—
0x1D	GS (group separator)	0x3D	=	0x5D	]	0x7D	}
0x1E	RS (record separator)	0x3E	>	0x5E	ˆ	0x7E	
0x1F	US (unit separator)	0x3F	?	0x5F	_	0x7F	DEL

Kommentarer: Den första kolumnen innehåller diverse kontrolltecken där de viktigaste tecknen är 0x0A (ny rad) och 0x0D (förflyttning av markören till början av raden).

Referenser

[1] *Laborationsmiljön Darma*, www.isy.liu.se/edu/kurs/TSEA28/laborationer

[2] *ARM Cortex-M4 Instruction Set*, infocenter.arm.com

[3] *ARM® Cortex®-M4F Based MCU TM4C123G LaunchPad^{TM} Evaluation Kit*, www.ti.com/tool/ek-tm4c123gxl

[4] *TM4C123GH6PM data sheet*, www.ti.com/product/TM4C123GH6PM

[5] *Code Composer Studio (CCS) Integrated Development Environment (IDE)*, www.ti.com/tool/ccstudio

Revisioner

2.0	Byte till ARM.
2.01	Byt från bx lr till mov pc,lr
2.02	Byt innehåll i printchar så det matchar lab1.asm
2.03	Korrigerat figur och testsekvens av printchar
2.04	Ändring av subrutineråterhopp tillbaks till bx lr, byt riktning på buffert
2.05	korrigera && till & för mov och movt i printchar
2.06	korrigera r4 till r0 för textbeskrivning av printchar
2.07	Korrigera extrauppgift, tryck på att subrutiner ska göras
3.0	Ändrat portar, lagt till stöd för lab på distans
3.01	Korrigerat register för printstring
3.2	Förtydligat krav på subrutiner
3.3	Kräv att subrutiner inte gör mer än beskrivet i beskrivningen
3.4	Nya namn på anslutning till tangentbord
3.5	Tydligare krav, utökat test för getkey