Embedded Systems 2 *P01*

Texas Instruments Code Composer™ Studio

Cortex-M4
Tiva™ TM4C1294

GPIO - ISR - Timer

Lernziele

- I/O Ports und Interrupts am Cortex-M4 *TIVA TM4C1294* Mikrocontroller aufsetzen
- Timer und Timer Interrupts korrekt verwenden
- ESP-I/Os über die `ESP_EKTM4C1294XL` Library verarbeiten
- Messungen mit Waveforms (Analog Discovery) durchführen

Einleitung

Infrastruktur, Tools, Hilfsmittel

	Doku, Manuals	**Tools**
①	CCS6_Readme	Embedded System Platform (ESP)
②	ESP_Readme	WaveForms (Analog Discovery)
③	EK-TM4C1294XL LaunchPad UserGuide	Code Composer Studio (CCS)
④	TIVA TM4C1294NCPDT DataSheet	CCS-Projects

Aufgabenstellung

Dieses Praktikum dient dazu, den Cortex-M4 *TIVA TM4C1294* Mikrocontroller und das zugehörige *EK-TM4C1294XL* LaunchPad kennen zu lernen.

Sie schreiben ein C-Programm, um einen Tastendruck am LaunchPad erkennen zu können und diese Aktion entsprechend auf den LEDs darzustellen. Die entsprechenden Port-Konfigurationen sind auf der Register-Ebene vorzunehmen. In einem weiteren Schritt werden sie die Interrupt-Funktionalität für die Taster umsetzen.

Sie erarbeiten anschliessend Schritt für Schritt ein HAL-Modul für die Timer-Peripherien des Cortex-M4 *TIVA TM4C1294* Mikrocontroller. Zudem werden Sie Messungen mittels Analog Discovery und WaveForms Tool durchführen, um die Registerkonfigurationen zu überprüfen.

Für das gesamte Praktikum arbeiten Sie mit dem LaunchPad. Es werden primär die auf dem LaunchPad zur Verfügung stehenden Taster (USR_SW1, USR_SW2) und LEDs (D1-D4) verwendet. Zusätzlich können Sie mittels der `ESP_EKTM4C1294XL` Library auch erweiterte Funktionalitäten der ESP nutzen.

Die nachfolgenden Aufgabenstellungen bauen alle auf dem bereitgestellten CCS-Projekt auf.

1 Pre-Lab

Machen Sie sich mit den Inhalten der Dokumente ①, ②, ③ und ④ vertraut, damit Sie während dem In-Lab effizient damit arbeiten können und die nötigen Informationen finden.

2 In-Lab

2.1 Theoriefragen zum TM4C1294NCPDT und dem zugehörigen LaunchPad

Um die nachfolgenden Theoriefragen beantworten zu können, müssen Sie sich vertieft mit den Dokumenten ③ und ④ auseinandersetzen.

2.1.1 Facts zum TM4C1294NCPDT

a) Was für ein Prozessor-Kern steckt im *TM4C1294NCPDT*?

b) Über wie viele General-Purpose In-/Output (GPIO) Blöcke (Ports) verfügt der Mikrocontroller?

c) Mit welcher maximalen Clock-Frequenz kann der Controller betrieben werden?

d) Welche Operationen werden von der Floating Point Unit (FPU) des Mikrocontrollers unterstützt?

e) Über welche(s) Register wird ein GPIO als Ein- bzw. Ausgang definiert (Registername)?

f) Über welche(s) Register wird der entsprechende GPIO-Zustand gelesen/geschrieben (Registername)?

g) Bei welchen Ports kann für jeden Pin ein separater Interrupt-Handler hinterlegt werden?

h) Was wird über die Pad Control Register konfiguriert?

2.1.2 Facts zum EK-TM4C1294XL LaunchPad

Benutzen Sie zur Beantwortung dieser Fragen das Schema im Dokument ③.

a) Auf welchen Ports befinden sich die LEDs D1-D4 auf dem LaunchPad?

b) Mit welchen Ports/Pins sind die beiden User Switches USR_SW1 und USR_SW2 verbunden?

2.1.3 Timer-Peripherie vom TM4C1294NCPDT

a) Was für Timer besitzt der *TM4C1294NCPDT*? Wie viele davon?

b) Auf welchen Pins des *TM4C1294NCPDT* kann der I/O „Timer5 Capture Compare 1“ gelegt werden?

c) Wie lautet die Interrupt-Vector Nummer für den Timer5_B (Capture Compare 1)?

2.2 GPIO-Ports konfigurieren

In dieser Aufgabe geht es darum, die Register zur GPIO-Konfiguration näher kennen zu lernen und deren Registerwerte zu setzen, damit die GPIOs korrekt verwendet werden können.

2.2.1 GPIO-Register

Für die folgenden Aufgaben werden die vier LEDs (D1-D4) auf dem LaunchPad und ein Taster (USR_SW1) benötigt. Die nachfolgende Übersicht zeigt, wie die entsprechenden Pins konfiguriert werden müssen.

Peripherie	Pin	Konfiguration
`LED_1 (D1)`	`PN1`	`Output, 2mA, Digital enable`
`LED_2 (D2)`	`PN0`	`Output, 2mA, Digital enable`
`LED_3 (D3)`	`PF4`	`Output, 8mA, Digital enable`
`LED_4 (D4)`	`PF0`	`Output, 8mA, Digital enable`
`Taster_1 (USR_SW1)`	`PJ0`	`Input, Pull-up/Pull-down?`

a) Wird für den User-Taster (`USR_SW1`) der Mikrocontroller-interne Pull-up oder Pull-down Widerstand der GPIO-Peripherie gebraucht? Falls ja, welcher?

b) Bevor die Register eines GPIO-Moduls beschrieben werden können, muss das Register `RCGCGPIO` konfiguriert werden.
Was genau wird mit diesem Register konfiguriert?
Mit welchem erforderlichen Bitmuster muss das Register konfiguriert werden?

	Register	Bitmuster	Wirkung
	SYSCTL_RCGCGPIO_R	%	

c) Wie lauten die Port_Base Adressen für die untenstehenden GPIO's?

Port	Port_Base Adresse [hex]
PORT F	
PORT J	
PORT N	

d) Wie lauten die Register-Offsets für die untenstehenden Register?

Register	Register_Offset [hex]
GPIODATA	0x000
GPIODIR	0x400
GPIODR2R	0x500
GPIODR8R	
GPIOPUR	
GPIOPDR	
GPIODEN	

e) Die untenstehende Tabelle enthält die erforderlichen Bitmustern, damit die LEDs (D1-D4) und der Taster (USR_SW1) auf dem LaunchPad richtig über die jeweiligen GPIOs angesprochen werden können. Studieren Sie die Wirkung der jeweiligen Register-Konfigurationen.

	Register	Bitmuster	Wirkung
	GPIO_PORTN_DIR_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxx11	Configure Bit0 and Bit1 on GPIO PortN as outputs
	GPIO_PORTN_DR2R_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxx11	Configure output currents up to 2mA on Bit0 and Bit1
	GPIO_PORTN_DATA_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxx00	Set output values to zero (LED OFF) on Bit0 and Bit1
	GPIO_PORTN_DEN_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxx11	Enable digital functionality on Bit0 and Bit1
	GPIO_PORTF_DIR_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxx1xxx1	Configure Bit0 and Bit4 on GPIO PortF as outputs
	GPIO_PORTF_DR8R_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxx1xxx1	Configure output currents up to 8mA on Bit0 and Bit4
	GPIO_PORTF_DATA_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxx0xxx0	Set output values to zero (LED OFF) on Bit0 and Bit4
	GPIO_PORTF_DEN_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxx1xxx1	Enable digital functionality on Bit0 and Bit4

	GPIO_PORTJ_DIR_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxxx0	Configure Bit0 on GPIO PortJ as input
	GPIO_PORTJ_PUR_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxxx1	Enable pull-up on Bit0
	GPIO_PORTJ_DEN_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxxx1	Enable digital functionality on Bit0

2.2.2 Register-Konfiguration umsetzen

Nehmen Sie nun die obigen Register-Konfigurationen vor. Dazu steht ein vorbereitetes Projekt zur Verfügung. Ergänzen sie die Funktion `hal_gpio_init()` in der Datei `hal_gpio.c` so, dass die GPIO-Module korrekt konfiguriert werden.

Überprüfen Sie die Konfiguration in CCS, indem Sie sich die entsprechenden Register im Debug-Modus anschauen (View Registers).

2.2.3 Taster-Status einlesen und auf LEDs ausgeben

Erweitern Sie im Modul `HAL_GPIO` die Funktionen `hal_gpio_read()` und `hal_gpio_write()` an den dazu vorgesehenen Stellen in der Datei `hal_gpio.c`.

Testen Sie das Programm anschliessend aus.

Innerhalb der Hauptschleife im `main.c` werden bereits die dazu notwendigen Funktionsaufrufe gemacht, so dass der Taster (USR_SW1) eingelesen wird. Dieser Zustand wird anschliessend auf den vier LEDs (D1-D4) signalisiert.

Folgende Tabelle zeigt die gewünschte Funktionalität:

Taster gedrückt	Alle LEDs on
Taster losgelassen	Alle LEDs off

2.3 GPIO-Interrupt konfigurieren

In dieser Aufgabe geht es nun darum, einen Tastendruck per Interrupt zu erkennen und über dessen Interrupt Service Routine (ISR) entsprechend die LEDs zu setzen oder zu löschen. Sie können dazu das in Aufgabe 2.2 erarbeitete Projekt erweitern.

Folgende Schritte müssen vorgenommen werden, um einen Tastendruck mittels Interrupt erkennen zu können um entsprechend zu reagieren (dieses Vorgehen gilt (oder ist ähnlich) für Interrupts im Allgemeinen):

1. GPIO Konfigurationen (in Aufgabe 2.2.2 Register-Konfiguration umsetzen bereits erledigt)
2. Interrupt Maske für den entsprechenden Port/Pin löschen
3. Interrupt Sense, both edge, event entsprechend der benötigten Anwendung konfigurieren
4. Interrupt Flag löschen für den entsprechenden Port/Pin
5. Interrupt Maske für den entsprechenden Port/Pin setzen
6. GPIO Interrupt beim NVIC einschalten

2.3.1 GPIO-Interrupt aufsetzen

a) Versuchen Sie, die untenstehende Tabelle mit den notwendigen Interrupt-Einstellungen nachzuvollziehen. Der Taster (USR_SW1) soll auf beide Flanken einen Interrupt generieren. Ergänzen Sie anschliessend die Funktion `hal_gpio_init()` in der Datei `hal_gpio.c`.

	Register	Bitmuster	Wirkung
	GPIO_PORTJ_IM_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxxx0	Clear interrupt mask (GPIOIM) on Bit0 (USR_SW1)
	GPIO_PORTJ_IS_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxxx1	Setup interrupt sense as edge detection (GPIOIS)
	GPIO_PORTJ_IBE_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxxx1	Set interrupt both edge (GPIOIBE)

	GPIO_PORTJ_IEV_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxxx0	Set falling edge as trigger for an interrupt (GPIOEV) Note: This register is not needed, because both edge are active!
	GPIO_PORTJ_ICR_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxxx1	Clear interrupt (GPIOICR) Note: For edge detect interrupts, writing a 1 to the IC bit in the GPIOICR register clears the corresponding bit in the GPIORIS and GPIOMIS registers
	GPIO_PORTJ_IM_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxxxx1	Unmask, enable the interrupt (GPIOIM) on Bit0 (USR_SW1)
	NVIC_EN1_R	%xxxxxxxx'xxxx1xxx'xxxxxxxx'xxxxxxxx	Enable GPIO PortJ interrupts on NVIC Note: GPIO PortJ vector number = 67, this corresponds to interrupt number = 51

2.3.2 ISR (Interrupt Service Routine)

Erstellen Sie eine Funktion, die als ISR agieren soll. Der zugehörige Funktionsprototyp muss folgendermassen aussehen:

```
void _hal_gpio_portJISR(void);
```

Damit diese Funktion nun effektiv beim entsprechenden Interrupt aufgerufen wird, muss auch die Interrupt-Vektortabelle entsprechend angepasst werden. Die Vektortabelle ist in der Datei `tm4c1294ncpdt_startup_ccs.c` zu finden.

Überprüfen Sie die eingetragenen Funktionen in der Tabelle.

Jetzt muss noch dafür gesorgt werden, dass der Linker die Adresse der ISR dann auch wirklich findet, bzw. auflösen kann. Dies wurde mit folgender extern Funktionsdeklaration gemacht:

```
extern void _hal_gpio_portJISR(void);
```

2.3.3 Interrupt auswerten

Der Taster (USR_SW1) liegt auf dem Port J. Da der Port J nur einen Port-Interrupt besitzt, wird für jeden möglichen Pin auf diesem Port die gleiche ISR aufgerufen. Dies ist bei vielen Mikrocontrollern so üblich. Daher muss in der ISR festgestellt werden, welcher Pin effektiv für das Auslösen des Interrupts verantwortlich ist. Benutzen Sie dazu das entsprechende Interrupt-Status-Register.

Die Applikation soll noch die genau gleiche Funktion haben wie bis anhin. Werten Sie dazu innerhalb der ISR den aktuellen Zustand des Tasters (USR_SW1) aus und setzen oder löschen Sie die LEDs (D1-D4) entsprechend.

Testen Sie ihr Programm aus.

Hinweis: *Vergessen Sie nicht, in der Hauptschleife des `main.c`, den nicht mehr benötigten Code zu deaktivieren (auskommentieren). Es soll nur noch mit Interrupts gearbeitet werden!*

2.4 General-Purpose TIMER konfigurieren

In dieser Aufgabe geht es nun darum, die Register zur TIMER-Konfiguration kennen zu lernen und die Inhalte so zu setzen, damit der TIMER als Intervall-Timer korrekt agiert.

Hinweis: *Für diesen Teil der Aufgabe soll nun anstatt `hal_gpio.c` die erweiterte Funktionalität von `hal_gpio_timer.c` verwendet werden. Tauschen Sie die Files im Projekt mittels CCS-Option „Exclude from Build" aus.*

2.4.1 Hal_timer_init()

Vervollständigen Sie im File `hal_timer.c` die Funktion `hal_timer_init()`, so dass die `Timer5_B` Peripherie als 16Bit Timer betrieben werden kann.

Dazu müssen Sie den Timer wie folgt konfigurieren:

1. Enable Clock für Timer5
2. Intervall-Timer im Up-Mode
3. Das Intervall soll aus dem msInterval-Parameter berechnet werden
   - Dazu muss der System-Clock mithilfe der ESP-API Funktion `esp_sysCtlClockGet()` ermittelt werden
   - Beachten Sie, dass die 16Bit knapp sind → machen Sie daher gebrauch vom Prescale-Register (hiermit kann der Zähler auf 24Bit erweitert werden)
4. Bei einem Timeout soll ein Interrupt generiert werden
   - Die ISR (`void _hal_timer_t5ccp1ISR(void)`) soll im Modul `hal_timer.c` realisiert werden
   - Die ISR soll die Callback-Funktion (pCallbackFct-Parameter der `hal_timer_init()`-Funktion) aufrufen
   - Die ISR selbst ist bereits in der Vektortabelle eingetragen

a) Versuchen Sie, die untenstehende Tabelle mit den erforderlichen Bitmustern, damit der Timer korrekt initialisiert wird, nachzuvollziehen.

	Register	Bitmuster	Wirkung
	SYSCTL_RCGCTIMER_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xx1xxxxx	Activate clock of Timer5 Timer5 is now ready for access
	TIMER5_CTL_R	%xxxxxxxx'xxxxxxxx'xxxxxxx0'xxxxxxxx	Disable Timer5B
	TIMER5_CFG_R	%00000000'00000000'00000000'00000000	Clear Timer configuration
	TIMER5_CFG_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxx1xx	Use Timer5 as two 16bit timers
	TIMER5_TBMR_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxx1xx1x	Setup timer with periodic mode up-counting (TimerB mode register)
	TIMER5_TBILR_R	interval & 0x0000FFFF	Writes upper bound for TimerB (Lower 16Bit of interval)
	TIMER5_TBPR_R	(interval >> 16) & 0x000000FF	Prescale (8Bit) value (Upper 8Bit of interval)
	TIMER5_IMR_R	%xxxxxxxx'xxxxxxxx'xxxxxxx1'xxxxxxxx	Set the interrupt mask register (TimerB timeout)
	NVIC_EN2_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'xxxxx1xx	Enable Timer5 interrupt on NVIC Note: Timer5 vector number = 82, this corresponds to interrupt number = 66

2.4.2 Hal_timer_start()

Vervollständigen Sie im File `hal_timer.c` die Funktion `hal_timer_start()`, sodass der `Timer5_B` zu zählen beginnt.

a) Ergänzen Sie die untenstehende Tabelle mit dem dafür erforderlichen Bitmuster.

	Register	Bitmuster	Wirkung
	TIMER5_CTL_R	%	

2.4.3 Hal_timer_stop()

Vervollständigen Sie im File `hal_timer.c` die Funktion `hal_timer_stop()`, so dass der `Timer5_B` nicht mehr zählt.

a) Vervollständigen Sie die untenstehende Tabelle mit dem dafür erforderlichen Bitmuster.

	Register	Bitmuster	Wirkung
	TIMER5_CTL_R	%	

Testen Sie ihre Registerkonfigurationen aus, indem Sie das vorgegebene Blink-Programm (100 ms Intervall) auf der Hardware ausführen. Wenn Ihre Einstellungen in Ordnung sind, sollte die LED4 alle 100 ms ihren Zustand ändern.

Hinweis: *Die Zeitdauer selbst können Sie allerdings erst nach dem Umsetzen der nächsten Aufgabe mit dem WaveForms messen.*

2.5 TIMER Output konfigurieren

Die Timer-Peripherie besitzt die Möglichkeit, den Zustand eines speziellen Pins (`T5CCP1`) bei jedem Timeout-Intervall des Timers, direkt in Hardware zu wechseln. Dazu soll der Pin7 des PortM entsprechend konfiguriert werden.

Dazu müssen die folgenden Schritte ausgeführt werden:

1. GPIO konfigurieren
   - Clock aktivieren
   - Pin als Ausgang definieren
   - Alternierende Funktion für diesen Pin
   - Port Multiplexer entsprechend konfigurieren
   - Pin als digital aktivieren
2. TIMER: Wechseln des Pin-Zustandes nach jedem Timeout

a) Versuchen Sie, die untenstehende Tabelle mit den dafür erforderlichen Bitmustern nachzuvollziehen.

	Register	Bitmuster	Wirkung
	SYSCTL_RCGCGPIO_R	%xxxxxxxx'xxxxxxxx'xxxx1xxx'xxxxxxxx	Activate Clock and Power of PortM. GPIO Ports are now ready for access.
	GPIO_PORTM_DIR_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'1xxxxxxx	Configure Bit7 on GPIO PortM as output
	GPIO_PORTM_AFSEL_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'1xxxxxxx	Select alternate function on Bit7
	GPIO_PORTM_PCTL_R	%xx11xxxx'xxxxxxxx'xxxxxxxx'xxxxxxxx	Setup port control (port mux), ④ S.788/1809
	GPIO_PORTM_DEN_R	%xxxxxxxx'xxxxxxxx'xxxxxxxx'1xxxxxxx	Enable digital functionality on Bit7
	TIMER5_TBMR_R	%xxxxxxxx'xxxxxxxx'xx1xxxxx'xxxxxxxx	Toggle state on timeout (TimerB mode register)

b) Erweitern Sie im File `hal_timer.c` die Funktion `hal_timer_init()` um diese Funktionalität.

Testen Sie Ihre Registerkonfigurationen aus, indem Sie das vorgegebene Blink-Programm (100 ms Intervall) auf der Hardware ausführen. Kontrollieren Sie zudem mit dem WaveForms Setting-File `06_LP_RIGHT_RGBLED_PWM`, ob das vorgegebene Zeitintervall korrekt ist.

3 Post-Lab

Vergleichen Sie Ihre eigenen Lösungen mit den bereitgestellten Musterlösungen.

Wo stellen Sie Unterschiede in der praktischen Umsetzung der Implementierungen fest? Beurteilen Sie die allenfalls unterschiedlichen Lösungsansätze und technischen Umsetzungen.