Themen: Digital I/O, Bitmanipulation, Polling und einfache Zeitmessungen
Letzte Abgabe: Am Anfang des dritten Praktikumstermins

Aufgabe[1] Bei einem rotierenden Gegenstand wird mit Hilfe eines inkrementellen Drehgebers der Drehwinkel ermittelt. Oft besteht ein Drehgeber aus einer Scheibe mit Schlitzen. Wenn sich die Scheibe dreht, wird abwechselnd der Weg für einen Lichtstrahl von einer LED zu einer Photodiode versperrt und wieder frei gegeben. Zur Ermittlung der Drehrichtung wird eine zweite Photodiode ein wenig versetzt angebracht (Bild 1).

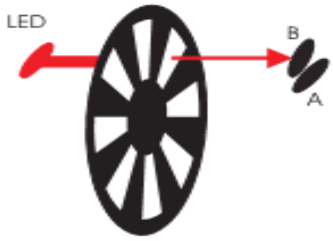


Bild 1: Drehgeber mit versetzten Photodioden

Solch eine Anordnung ist im TI-Labor als Lehrmodell mit einer Drehscheibe aufgebaut (s. Bild 2). In MS-Teams liegt das Datenblatt des Drehgebers vor. Für die Entwicklung zuhause werden die Signale des Drehgebers durch ein Programm auf dem Raspberry Pi Pico nachgebildet/simuliert. Details dazu finden Sie in MS-Teams**.**

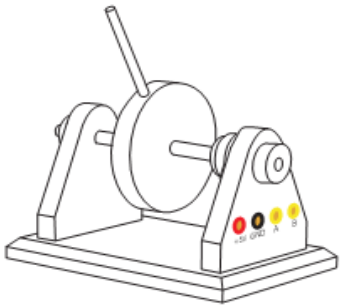


Bild 2: Drehgeber im Labor

Die Werte der Dioden sind auf die Kanäle A und B abgebildet. Wie Bild 3 zeigt, lassen sich die Impulsverläufe an Kanal A und B in vier Phasen a, b, c und d einteilen: Bei einer Vorwärtsbewegung werden die Phasen zyklisch in der Reihenfolge a, b, c und d durchlaufen. Bei einer Rückwärtsbewegung werden die Phasen in umgekehrter Reihenfolge durchlaufen. Durch Beobachtung der Phasen lässt sich die Bewegungsrichtung der Scheibe und die Anzahl der durchlaufenen Phasen feststellen. Ist die aktuelle Phase b so ergeben sich bei der nächsten Beobachtung der Signale die Fälle aus Tabelle 1.

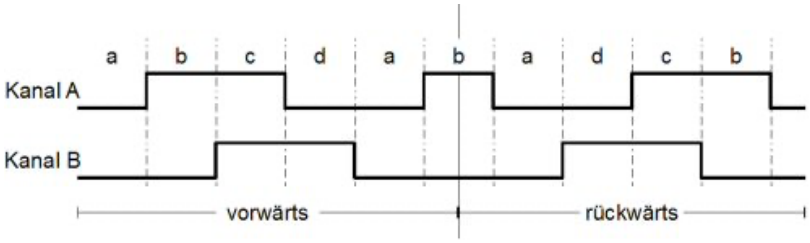


Bild 3: Ausgangssignale des Drehgebers in Abhängigkeit des Drehwinkels

Zuordnung Signale - Phase		
Kanal A	Kanal B	Phase
0	0	a
1	0	b
1	1	c
0	1	d

Zuordnung Phasenwechsel - Ereignis	
Phasenwechsel	Ergebnis
Phase b →Phase a	Rückwärtslauf
Phase b →Phase b	Keine Änderung
Phase b →Phase c	Vorwärtslauf
Phase b →Phase d	Fehler

Tabelle 1: Phasen und Phasenübergänge

[1] Diese Aufgabe basiert auf Unterlagen zum GSP Praktikum von Prof. Dr. Heiner Heitmann, HAW Hamburg.

Anschluss an das ITS-Board Der Drehgeber bzw. der Pi Pico (zur Simulation des Drehgebers) wird über die Pins IN0 (GPIO PF0) und IN1 (GPIO PF1) mit den ITS-Board verbunden. Beachten Sie bei der Nutzung des Drehgebermodells im Labor, dass der Drehgeber mit **3,3V** über das ITS-Board versorgt wird (s. Bild der Verkabelung in MS-Teams).
Die zu IN0 und IN1 gehörigen LEDs zeigen den Status der Pins / Photodioden an. Bei langsamen Drehen erkennt man an den LEDs die einzelnen Phasen und kann somit schnell überprüfen, ob die Hardware korrekt angeschlossen ist.
Mit Blick auf die Qualitätssicherung Ihres Programms ist der Drehgeber ungenau. Bitte nutzen Sie den auf dem Pi Pico implementierten Rechteckgenerator zum Test Ihres Programms (s. MS-Teams). **Während der Abnahme der Aufgabe werden Drehgeber und Pi Pico verwendet**.

Funktionsumfang des Programms

- Unter Beachtung der Drehrichtung wird die Anzahl der Schritte / Phasenwechsel - nicht der Winkel in Grad - binär codiert an den LEDs D8 bis D15 ausgegeben. Sie sind an den GPIOs PD0 bis PD7 angeschlossen. Es werden nur die niederwertigsten 8 Bits dieses Vorwärts-Rückwärtszählers angezeigt.
- Auf dem LCD-Display werden der Drehwinkel in Grad und die Drehgeschwindigkeit / Winkelgeschwindigkeit in Grad pro Sekunde ausgegeben. Der Winkel muss exakt angegeben werden, die Winkelgeschwindigkeit darf im Nachkommabereich abweichen. Zur Bestimmung der Winkelgeschwindigkeit benötigen Sie einen Timer zur Zeitmessung (s. timer Modul der ITS_BRD_LIB Bibliothek).
- LED D23 leuchtet, wenn als letztes eine Drehung in Vorwärtsrichtung erkannt wurde.
- LED D22 leuchtet, wenn als letztes eine Drehung in Rückwärtsrichtung erkannt wurde.
- Im Fehlerfall leuchtet LED D21. Sie wird durch Löschen des Fehlers zurückgesetzt (s.u. Taster S6).
- Mit der Taste S6 wird der Fehlerzustand gelöscht.
- Führen Sie die Winkel- und Winkelgeschwindigkeitsberechnung mit dem Typ *double* aus.
- Das Direct Digital Control (DDC) Konzept aus der Vorlesung **muss** verwendet werden.

Analyse des Zeitverhaltens Bei „schnellem“ Drehen des Drehgebers erkennt Ihr Programm nicht alle Phasenwechsel und meldet in der Regel einen Fehler. In diesem Fall ist die Ausführung der Software zwischen dem Auslesen aufeinanderfolgender Phasen langsamer als der Phasenwechsel selbst. Aufgabe 5 wird dieses Problem mit Hilfe von Interrupts signifikant lindern.
Aufgrund des verwendeten Bussystems ist die Kommunikation zwischen LCD-Display und CPU und somit die Ausgabe auf dem Display sehr langsam. Der Zeitbedarf der Ausgabe auf dem LCD-Display muss ausgemessen **und dokumentiert** werden**.** Vergleichen **und dokumentieren** Sie diesen mit der Ausführungszeit eines Super-Loop Durchlaufs.
Führen Sie im Labor die Zeitmessungen mit dem Oszilloskop durch – ein einfaches Video zur Benutzung des Oszilloskops finden Sie unter MS-Teams. Zur Messung schließen Sie einen freien GPIO Pin des ITS-Boards an einen Kanal des Oszilloskops an (Anschluss der Masse nicht vergessen). Setzen Sie dann vor der Ausgabe auf dem Display den GPIO Pin auf high. Ist die Ausgabe abgeschlossen, treiben Sie den Pin wieder mit low. Die mit dem Oszilloskop gemessene

Impulsbreite entspricht (bis auf sehr kleine Abweichungen) der Ausführungszeit auf dem Display.

Durch das Zeitverhalten bedingte Anforderungen an Ihr Programm

- Aufgrund der langsamen Displayausgabe werden in der Super-Loop nur Winkel und Winkelgeschwindigkeit ausgegeben. Bildschirmtexte, die sich nicht ändern, werden nur einmalig in der Startphase ausgegeben und nicht überschrieben.
- Es ist sinnvoll, dass die Display-Ausgabe an die Anforderungen des Anwenders angepasst wird. Die Ausgabe sollte alle 250ms bis 500ms aktualisiert werden, wenn Winkel oder Winkelgeschwindigkeit sich geändert haben. Somit müssen Sie auch nur alle 250ms bis 500ms Winkel und Winkelgeschwindigkeit auf Basis der Anzahl der Phasenwechsel berechnen.
- Eine einfache Berechnung der Winkelgeschwindigkeit zählt in einem vorgegebenen Zeitfenster die Anzahl der Phasenwechsel und berechnet aus diesen Daten die Winkelgeschwindigkeit. Somit sollten die Zeitstempel, die Anfang und Ende des Zeitfensters festlegen, möglichst direkt hinter dem Zeitpunkt eines Phasenwechsels liegen. Da Winkel und Winkelgeschwindigkeit alle 250ms bis 500ms ausgegeben werden müssen, liegt die Breite des Zeitfensters zur Berechnung der Winkelgeschwindigkeit auch zwischen 250ms und 500ms. Nachdem 250ms des Zeitfensters vergangen sind, sollte Ihr Programm in jedem Super-Loop Durchlauf schauen, ob ein Phasenwechsel vorliegt und dann den Zeitstempel für das Ende des Zeitfensters nehmen. Nachdem 500ms vergangen sind, wird das Zeitfenster geschlossen und die Berechnung von Winkel und Winkelgeschwindigkeit auf jeden Fall ausgeführt.
- Wenn Ihre Lösung stabil läuft, führen Sie bitte folgende Optimierung durch: Die längste Zeitspanne, die zwischen zwei Einlesevorgängen der Kanäle A und B liegt, definiert im Prinzip die maximal mögliche Winkelgeschwindigkeit, die Ihr Programm verarbeiten kann. Da die Kanäle A und B einmal in der Super Loop gelesen werden, definiert somit die maximale Durchlaufzeit der Super Loop die maximal mögliche Winkelgeschwindigkeit, die Ihr Programm verarbeiten kann. Die Durchlaufzeit der Super Loop ist besonders groß, wenn Winkel und Winkelgeschwindigkeit ausgegeben werden. Also müssen Sie dies Zeit reduzieren – alternativ kann man mit Interrupts arbeiten (s. Aufgabe 5), was in der Regel die bessere Lösung ist. Da Sie Interrupts noch nicht beherrschen, gibt es folgende einfache Lösung, die die mögliche maximale Winkelgeschwindigkeit signifikant verbessert. Die Ausgabe auf dem Display braucht die meiste Zeit. Weiterhin ist die Ausgabe von n Zeichen zirka n-mal so lange wie die Ausgabe eines Zeichens. Also geben Sie nicht alle Zeichen in einem Durchlauf der Super Loop aus, sondern wenn eine Ausgabe ansteht in jedem Super Loop Durchlauf nur ein Zeichen. Das kann man relativ einfach umsetzen, in dem die Ausgabe in einen Buffer geschrieben wird. Wird in einem Durchlauf der Super Loop festgestellt, dass ein neuer Wert im Puffer vorliegt, wird der Puffer zeichenweise ausgegeben – pro Super Loop Durchlauf ein Zeichen.

Ein schriftliches Konzept muss spätestens am Abend vor dem zweiten Praktikumstermin im git Repo auf dem Server hochgeladen und mit einem aussagekräftigen Bezeichner „getagged“ sein. Dieses Konzept muss folgende Themen beinhalten:

- Sinnvolles Modulkonzept
- Auflistung der möglichen Eingabezustände und die Reaktion auf diese
- Konzept zur einheitlichen Behandlung von Fehlerzuständen

Abnahme der Aufgabe im Praktikum

- Das Konzept muss vorliegen.
- Kommentierter Quellcode, der dem C-Coding Style entspricht, muss vorliegen.
- Eine Liste der von Ihnen durchgeführten Tests - zusammen mit den Testergebnissen - muss vorliegen. Überlegen Sie sich sinnvolle Testfälle.
- Die oben geforderte Analyse des Zeitverhaltens muss vorliegen.
- Sie müssen Ihr Programm erklären können.

Hinweise

- In dieser Aufgabe ist bei der Erstellung des Modulkonzepts, seiner Schnittstellen und seiner Funktionen das Design Prinzip Separation of Concerns sehr wichtig, damit Sie Überblick und Kontrolle über Ihr Programm behalten. Teilen Sie anhand des Modulkonzepts den Programmieraufwand im Team auf und diskutieren Sie anschließend den Code.
- Für die Winkelberechnung benötigen Sie die Anzahl der Schlitze der Scheibe des Drehgebers. Der Drehgeber hat 300 Schlitze – also 1200 Phasenwechsel pro Umdrehung.
- Der Zugriff auf I/O erfolgt über Bit Manipulationen. Gehen Sie hier nach einem einheitlichen Verfahren vor (s. Vorlesung).
- Sie müssen den Zustand des Drehgebers kodieren. Der Standardweg ist eine Finite State Machine (FSM). Wenn Sie FSMs schon kennen, wenden Sie diese gerne an. Ansonsten kann man den Zustand des Drehgebers auch ohne FSM gut umsetzen, indem Fehlerfall, Drehrichtung (rechts, links, nicht bekannt) und aktuelle Phase in statischen Variablen des entsprechenden Moduls gespeichert werden. Bei beiden Vorgehensweisen speichert eine Variable die Anzahl der Phasenwechsel, wobei die Drehrichtung beachtet wird.
- Berechnen Sie Zeitfenster mit dem *timer* Moduls der ITS_BRD_LIB. Es stellt einen HW Timer so ein, dass das 32 Bit CNT Register 90 mal in einer Mikrosekunde um 1 inkrementiert wird. Hat das Register den Wert 0xFFFFFFFF erreicht, dann findet ein Überlauf statt und der nächste Registerwert ist 0x00000000. Die Differenz zweier Werte, die aus dem Timer Register gelesen wurden, entsprechen der Zeitspanne zwischen den beiden lesenden Zugriffen auf das Register.

Submodule:

State

rotate_left()

rotate_right()

rotate_no()

check_phase_change()

phase_shifter()

get_phase_now()

check_State()

Calc

rotation_speed()

Slits

slits_per_rotation()

angle_calc()

angle_speed()

Inputhandler

channel_reader()

angel_reader()

Button_check()

Outputhandler

print()

LED_Error()

LED_rotate()

timer

time_now()

time_before()

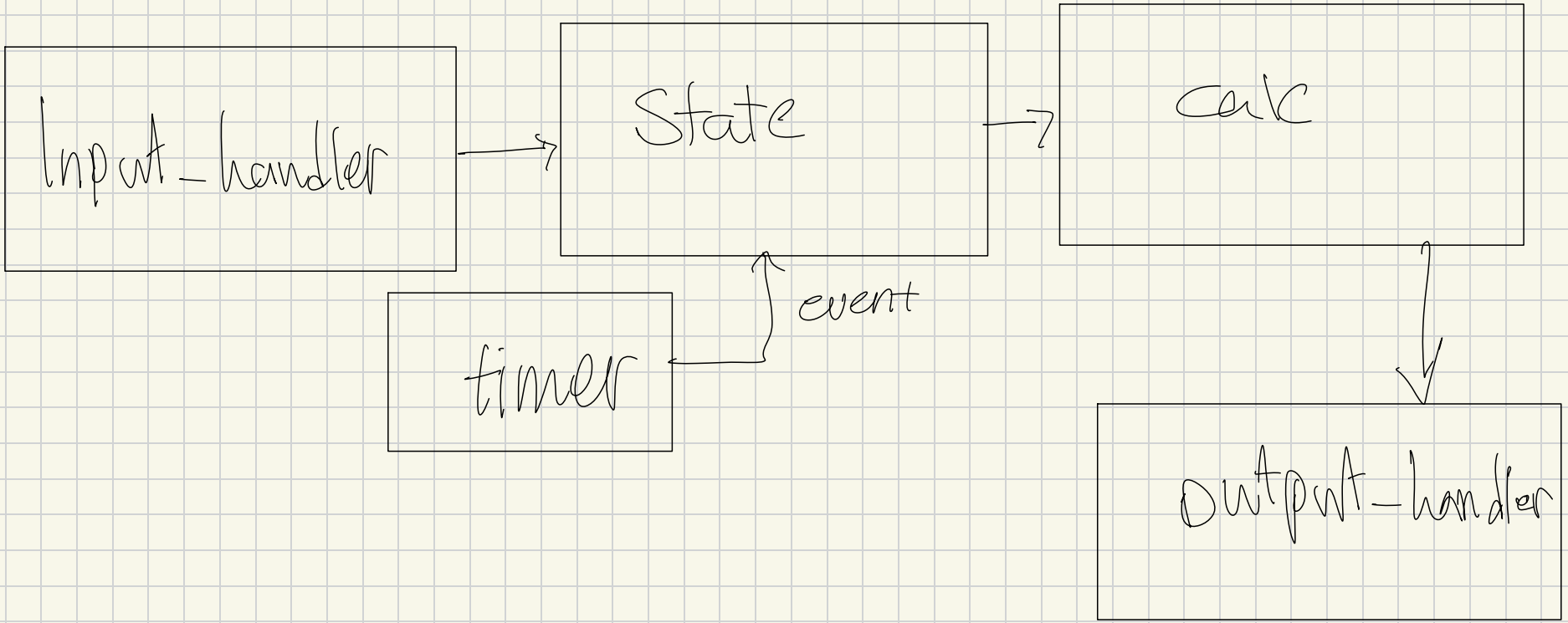
check_time_change()

version 0

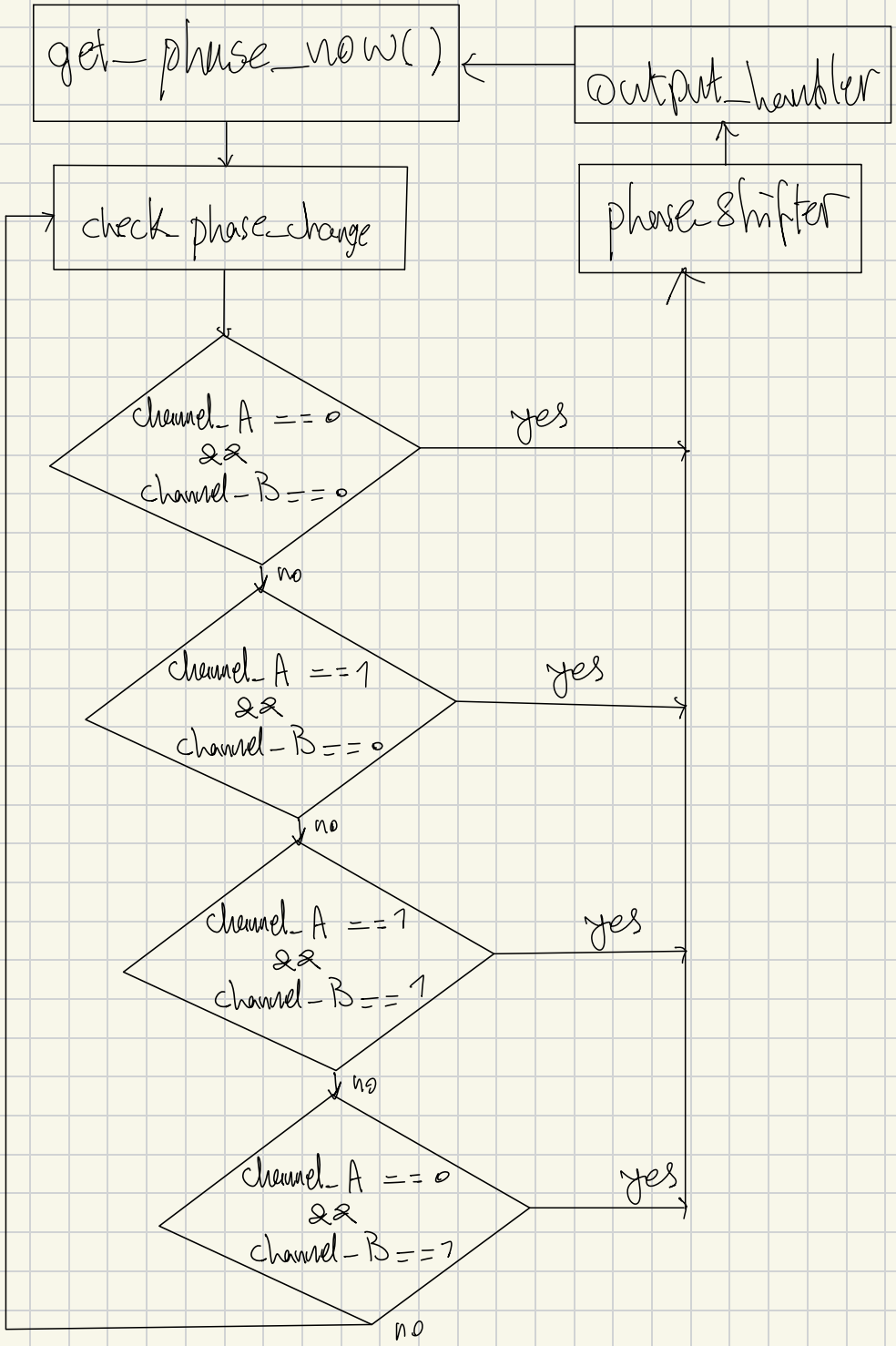
Version 0

DDC:

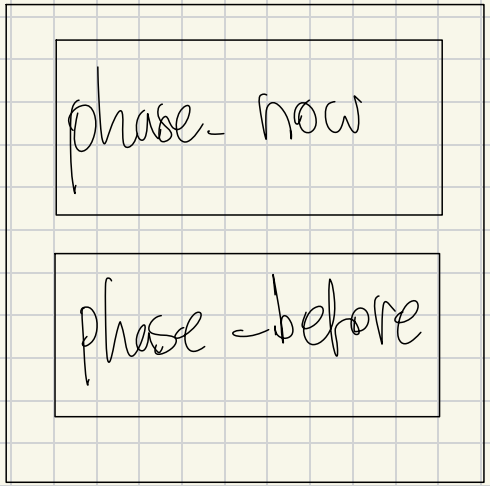
Es ist entscheidend, dass am Anfang der Super Loop alle Input Daten eingelesen werden und der Rest der Super Loop auf den eingelesenen Daten und nicht den In-Ports arbeitet.



State

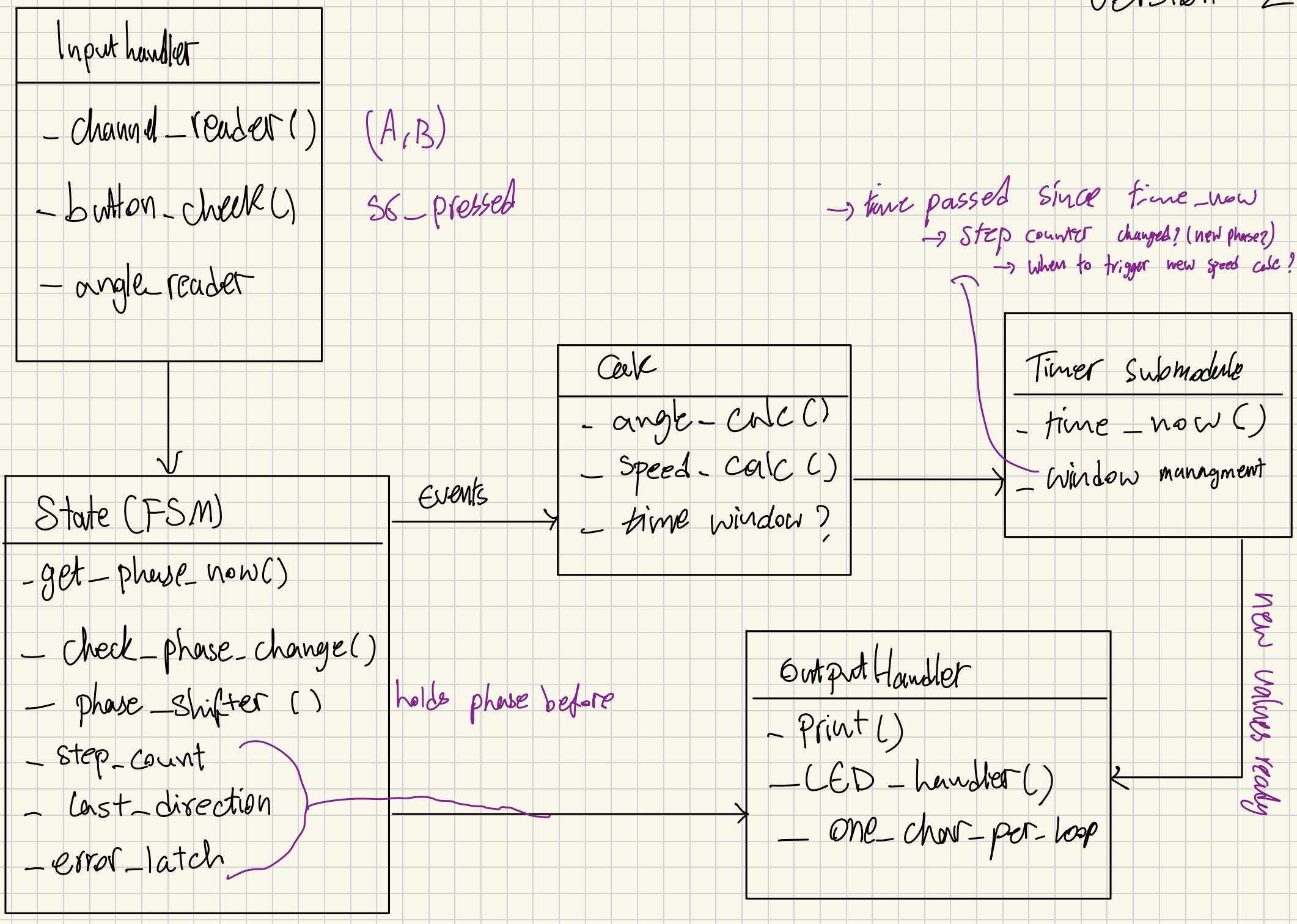


?



version 1

Version 2


Input handler
- channel_reader()
- button_check()
- angle_reader
(A,B)
SG_pressed
Calc
- angle_calc()
- speed_calc()
- time window?
Timer Submodule
- time_now()
- window managment
→ time passed since time_now
→ step counter changed? (new phase?)
→ when to trigger new speed calc?
State (FSM)
- get_phase_now()
- check_phase_change()
- phase_shifter()
- step_count
- last_direction
- error_latch
Events
holds phase before
OutputHandler
- Print()
- LED_handler()
- one_char_per_loop
new values ready

Timer

CLK_PSC → Prescaler → (f) → Counter

1st prescaler 2 $f = CLK\_PSC / 2$

Zeit	Befehl	CNT val
t_1	read CNT	v_1
t_2	read CNT	v_2

$\Delta_t = t_2 - t_1 = (v_2 - v_1)/f$

t_start
t_end
steps - start
steps_now
(360/1200)

$speed = (steps\_now - steps\_start)/(t\_end - t\_start) \cdot (360/1200)$

version 1

version 1

$\{-3, -2, -1, 0, 1, 2, 3\}$

delta = (now_index − prev_index) mod 4

a = 00 ⇒ 0

b = 10 ⇒ 1

c = 11 ⇒ 2

d = 01 ⇒ 3

⇒ normalizing to $\{-2, -1, 0, 1, 2\}$ by ± 4

+1	stepFwd
−1	stepBck
0	NoChange
±2	Error

or we do Gray-coding?

Version 2

Phase	channel A	channel B	Binary	Decimal
a	0	0	00	0
b	1	0	10	2
c	1	1	11	3
d	0	1	01	1

=> two Bit pattern

=> only one bit change at a time

Gray code

code_before = (A_before << 1) | B_before } val {0, 1, 2, 3}

code_now = (A_now << 1) | B_now

Transition = (code_before << 2) | code_now } val {0, 1, 2, 3, ..., 14, 15}

Transition (binary)	meaning	Hexdeci
0000, 0101, 1010, 1111	no change	0x00, 0x05 0x0A, 0x0F
0001, 0111, 1011, 1100	+1 step (forward)	0x01, 0x07 0x0B, 0x0C
0010, 0100, 1000, 1110	-1 step (Backward)	0x02, 0x04 0x08, 0x0E
anything else	invalid (Error)	everything else

Pros:

no mod 4 needed

readable

compact

replaces delta

output of this gives the event