0.1 SPI

The SPI protocol is a shift register protocol, with a possible register size of 8-16 bits. The communication protocol is designed as to keep size of the datagram less or equal to the register size, as to keep communication protocol between the FPGA and Tiva simple and utilise the full duplex capabilities of the SPI protocol.
The communication protocol uses the master-slave principle, with the Tiva as the master and the FPGA as slave.
When the Tiva initiates communication, Slave Enable is driven low and 1 SPI clock cycle later data will be read on rising edge from the SPI clock, as seen in figure 1.
Each datagram is 16 bits in size and the datagram format send by the Tiva can be seen in table 2 and the package format send by the FPGA can be seen in table 4. In the package format the bit number indicates the order of transmission starting with 0.

ref

Hvordan refererer vi til Tiva

Data	**Data type**
PWM	9 bit signed
Position	12 bit signed
Velocity	12 bit signed
Amps	12 bit unsigned
Home Index	1 bit

Table 1: *Data types*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PWM - 9bits									MS	Resp. select		RS	Reserved		

Table 2: *Package format - Tiva*

MS	Motor Select	
Response select		
	Position	00
	Velocity	01
	Acceleration	10
	Amps	11
RS	Reset Position	

Table 3: *Package format extended*

0.2 Controller design

0.2.1 Ziegler-Nichols

By using the Ziegler-Nichols approach the system is approximated to two second order linear systems which can be repsresented in the s-domain shown i eq. (0.1), where A is the final value of the stepresponse, τ is the rise time and t_d is the lag time. The lag time is assumed to be 5ms as that is the fastest the the system can possibly update due tot he systick timer in freeRTOS .

ref free rtos

Table 4: *Package format - FPGA*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Res		HS1	HS0	Requested Data											

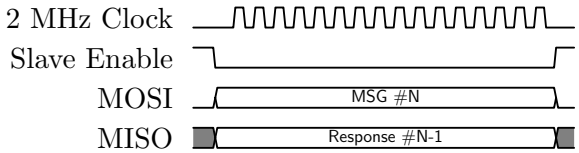

Figure 1: *SPI timing diagram*

$$\frac{A}{\tau s + 1} e^{st_d} \tag{0.1}$$

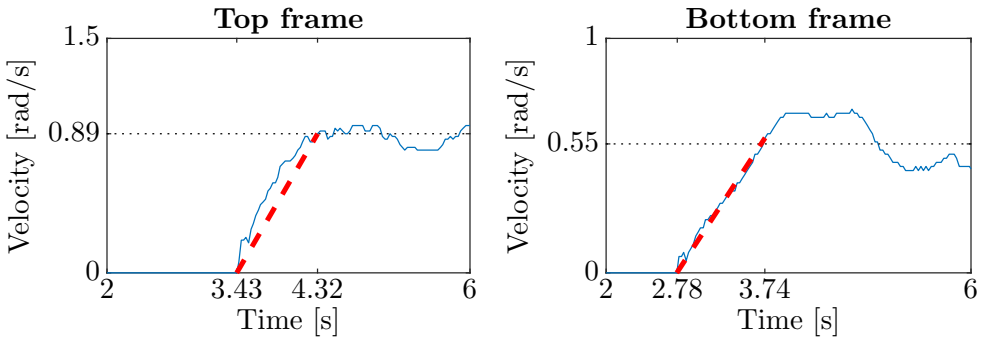

Figure 2: *Stepresponse with marked values for Ziegler-Nichols*

The top frame approximated to the S-domain representation in eq. (0.3) and the bottom frame in eq. (0.5)

$$G_{Top}(s) = \frac{0.89}{(4.32 - 3.43)\, s + 1} \tag{0.2}$$

$$= \frac{0.89}{0.89s + 1} \tag{0.3}$$

$$G_{Bottom}(s) = \frac{0.55}{(3.74 - 2.78)\, s + 1} \tag{0.4}$$

$$= \frac{0.55}{0.96s + 1} \tag{0.5}$$

A first estimate of the P and I terms are shown in eq. (0.7) and eq. (0.8) in accordance with the Ziegler-Nichols approach shown in eq. (0.6).

$$R = \frac{A}{\tau} \qquad K_p = \frac{0.9}{R\,L} \qquad T_i = \frac{L}{0.3} \qquad L = 5\text{ms} \tag{0.6}$$

$$K_{p,top} = 180 \qquad K_{i,top} = 0.02 \tag{0.7}$$
$$K_{p,bottom} = 314.18 \qquad K_{i,bottom} = 0.02 \tag{0.8}$$

Tilføj noget sca
ing

0.2.2 PI

0.2.3 Statespace

0.3 PWM

In this section the implementation and calculations of PWM[1] generation is described. PWM is a way of controlling the speed of DC-motors. It work by varying the time a given voltage is applied to the motor, relative to the time it is not. This relationship is called the duty cycle of the PWM signal. The duty cycle is depicted in figure 3.

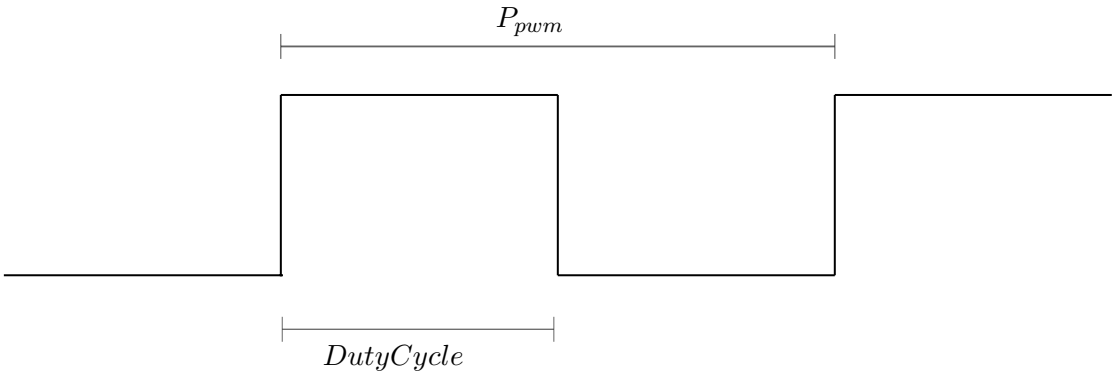

Figure 3: *Illustration of PWM signal. P_{pwm} is the period of PWM signal.*

Er der korrekt
er dutycycle ik
forholdet melle
de to, ikke kun
den dene perio
?

The duty cycle can be varied from $0-100\%$. Translating to a DC-motor torque of $0-max$. The PWM signal is generated using the FPGA, which is a hard requirement from the report specification. To generate a PWM signal as in figure 3 a counter in the FPGA is used. The output signal is set high, as soon as the counter starts counting. When the counter reaches the duty cycle value, the output signal is set LOW. When the counter then reaches its precalculated maximum value, the process starts over. Thus generating a PWM output. The maximum counter value is calculated based on the wanted frequency of the PWM signal. The frequency of the signal is important for a correct operation of the DC-motor, and depends on specific parameters of the motor. The optimal frequency of the motors in this project is calculated in the following section.

project?

on?

[1] Pulse Width Modulation

0.3.1 Determining the PWM frequency

A DC-motor can be modeled by the equivalent circuit depicted in figure 4.

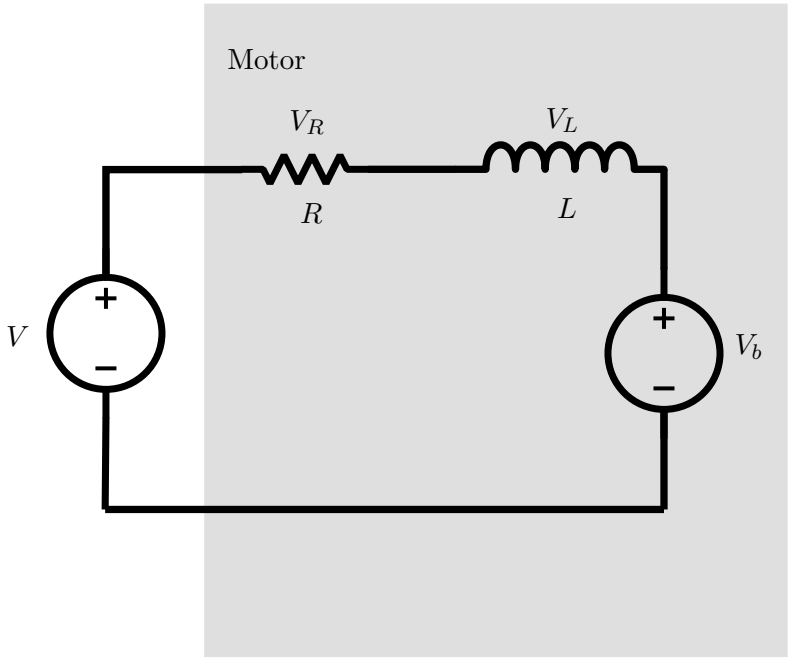

Figure 4: *Electrically equivalent DC-motor circuit.*

The circuit in figure 4 can be described with equation 0.10. With the parameters R resistance in Ohms, L inductance in Henry and V_b the back EMF generated by the motor when it is at speed.

$$V(t) = V_R(t) + V_L(t) + b\dot{\theta} \tag{0.9}$$

$$V(t) = V_R(t) + V_L(t) + V_b \tag{0.10}$$

$$\Rightarrow V_b = b\dot{\theta} \tag{0.11}$$

To find the maximum PWM frequency though, it is not necessary to take into account the back emf. This is because the highest PWM frequency possible, is at the exact moment, when the motor starts to move. Therefore a simplified circuit will be used, depicted in figure 5

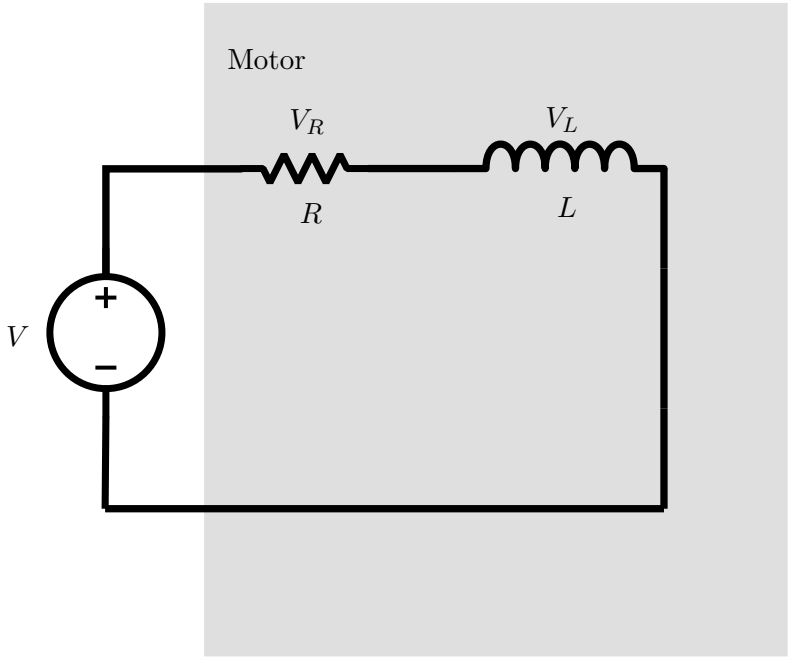

Figure 5: *Simplified electrically circuit of DC-motor*

In a DC-motor the torque of the motor is proportional to the current in the winding's of the motor. The motor winding's have an inductance L. If the motor is supplied by a constant source, the current through the resistance R and the winding's with inductance L is constant, and therefore the torque is constant. This is because a constant voltage over an inductor is equal to a short circuit over the inductor. In a system with a PWM source though, the voltage is not constant, and therefore the current through the inductor is not constant.
When choosing a PWM frequency it is therefore necessary to take the changing current in the inductor into account.
The voltage over the motor in figure 5 can be summed using Kirchoffs Voltage Law.

$$V(t) = V_R(t) + V_L(t) \tag{0.12}$$

$$V(t) = R \cdot i + L \cdot \frac{di}{dt} \tag{0.13}$$

The current $i(t)$ can be found by solving differential equation(0.13).

$$i(t) = \frac{V}{R} \cdot (1 - e^{\frac{R}{L} \cdot t}) \tag{0.14}$$

The time constant of equation 0.14 is $\tau = \frac{L}{R}$. As the current flows, the voltage over L becomes smaller, and the voltage over R becomes larger.Only when the voltage $V_L = 0$ the motor draws all of the available current. This implies that at least 5τ must have passed before the output of the PWM signal switches low, otherwise the motor will never draw full current, and thereby it will loose torque. This can be seen in figure 6

will loose *available* torque

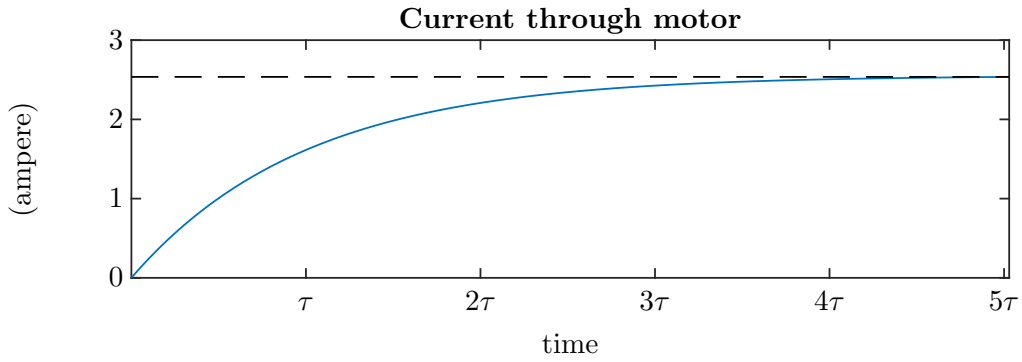

Figure 6: *Plotting the current over time when voltage is supplied to the DC-motor.*

This implies that the minimum on-pulse of the PWM signal must be at least $5\tau = 5 \cdot \frac{L}{R} = 4.9707 \cdot 10^{-4}s \approx 497\mu s$
The system has a large amount of friction in the motors and gears. This means that the system will not move with a duty cycle below 50%. The minimum PWM period is therefore $497\mu s \cdot 2 = 994\mu s$. The maximum frequency is $1/997\mu s = 1003Hz$. In figure 7 the outcome of a PWM frequency of $1003Hz$ can be seen. In the figure, the voltage is not as in the real system, which is $12V$ here it is the same as the peak current, to gave at better picture. The main point is that the current reaches peak value, before the voltage drops.

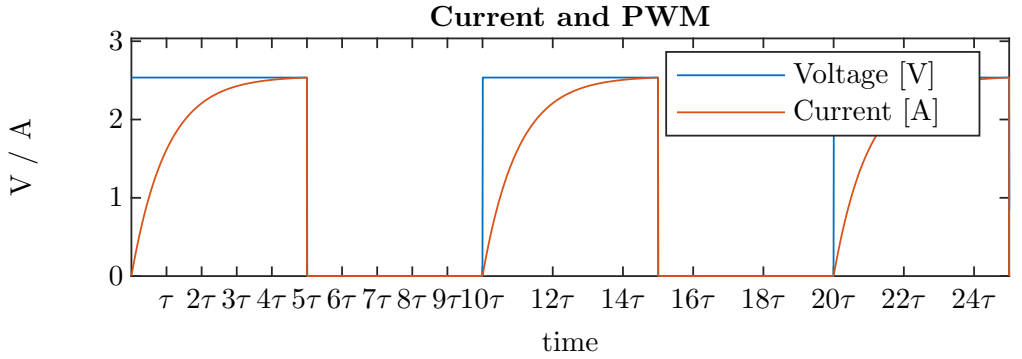

Figure 7: *Plotting the current through the motor, when voltage is applied.*

0.4 Quadrature Decoder

0.4.1 Quadrature Encoder

In order to understand the design of the Quadrature Decoder a small explanation of how a Quadrature Encoder works is needed.
A Quadrature Encoder uses two channels to sense the position and direction of, typically, a rotating disk/shaft or a linear strip. The disk or strip has two paths on it, positioned 90 degresse out of phase with each other, see figure 8 for a rotary encoder and figure 9 for a strip encoder. Statinoary sensores are typically placed on top of the encoder, so when a track moves in relation to a sensor, it outputs a logic high or low output signal depending on what part of the track is visible to the sensor. An encoder has two output signals, one for each channel, typically called A and B, see figure 10 for a representation of those for

Placeres sensorere oven pp encoder eller er det en del af d færdige encode

both encoder types. This two signals, A and B, is what allows the encoder to determine the position and direction of the encoder

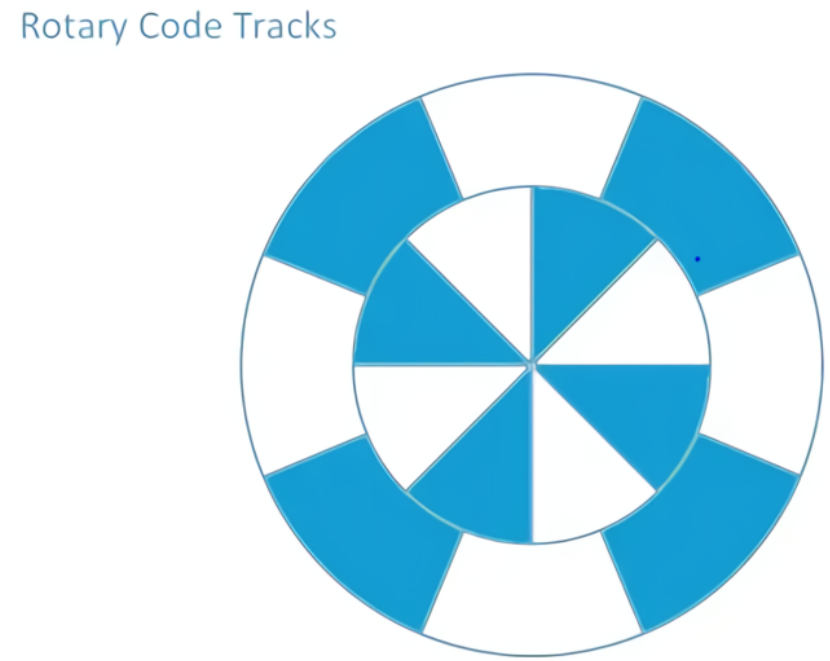

Figure 8: *Tracks of a rotary encoder.*

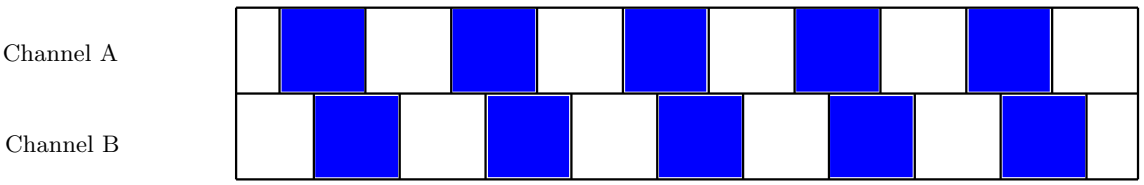

Figure 9: *Tracks of a Linear encoder*

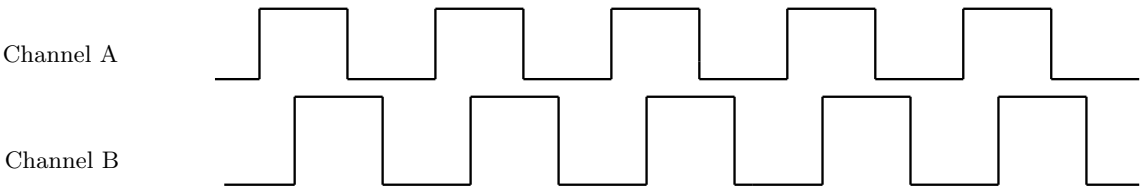

Figure 10: *The logic output of an encoder*

0.4.2 Decoder principle

The founding principle of a Quadratur decoder relies on decoding the relationship between the two outputs and deduce a position change. The idea is to observe both encoder outputs. By counting the transitions from high to low and low to high on just one of the outputs, it can be determined how far the encoder has rotated. However, by adding the second

output, the direction of the encoder can be computed as well as the resolution to the position. The encoder used in the project is a rotary encoder with a resolution of 360[2], which means that the Quadrature Decoder will count 360 "ticks" per shaft turn. This resolution and rotation can not be mapped directly to the system itself as its motors are geared. The motors do three full rotations of their shafts before the system itself has completed a full rotation. Hence the resolution for the system is 1080 ticks. This gives a $\frac{1}{3}$ of a degrees position precision, allowing a very precice measurement.

0.4.3 Statemachine

To use the transitions between encoder output A and output B, a truth table of every possible combination of the two singals must be computed. It is necessary to include a third signal called reset to allow for resetting of the position counter when a new home position is desired. The truth table computed from all three signals is shown in figure 11.

A_prev	A_new	B_prev	B_new	Reset	Direction	Position
0	1	0	0	1	Forward	+ 1
1	1	0	1	1	Forward	+ 1
1	0	1	1	1	Forward	+ 1
0	0	1	0	1	Forward	+ 1
0	0	0	1	1	Backward	- 1
1	0	0	0	1	Backward	- 1
0	1	1	1	1	Backward	- 1
1	1	1	0	1	Backward	- 1
x	x	x	x	0	No change	0

Figure 11: *This figure shows the truth table for the Quadrature decoder*

The truth table shows nine possible scenarios that can occur from tracking the transistions of the outputs. Four of these results in a forward direction and the counter will increment, four scenarios results in a backwards direction and the counter will decrement. The last scenario occurs if the reset flag is set low[3], and the counter is set to zero regardless of what it was on before. From these observations a state machine containing 6 states has been computed. The states are: AB_low with the gray code 001[4], AB_high with gray code 111, A_high with gray code 101, B_high with gray code 011, undefined and reset $xx0$. The undefined state is purely for debugging purposes used if an error with the signal A and B ocurs. A potential error could be if the state is currently AB_low the gray code can not be 111 in the next instance. This would mean that both signals changed at the same time. The state machine diagram is shown below in figure **??**.
The decoder works by first initialalizing and go to a state depending on the level of signal A and B. The decoder will then use the debounced signals to determine the next state. If the starting state is AB_low, there is three correct outcomes: the reset signal goes low and the counter will go to zero, the signal A goes high and the state changes to A_high or the last possible outcome, that does not give an error, is B_high. By looking at the truth table from the above figure 11, it can be observed that if the state goes from AB_low to A_high, the direction will be forward and the counter will increment where the counter would decrement and direction set to backwards if the next state is B_high.

[2]See datasheet x
[3]Reset is implemented as active low, as can be observed in the truth table 11.
[4]The first digit is signal A, second is signal B and thrid is the reset signal

The counter keeps track of the position of the encoder. The counter counts position zero as the angle the system was in, as the encoder was initialized. It is thereforth considered useful to add a reset function, so a new home position can be added as desired. The Statemachine approach has been implemented and the process described in section **??**

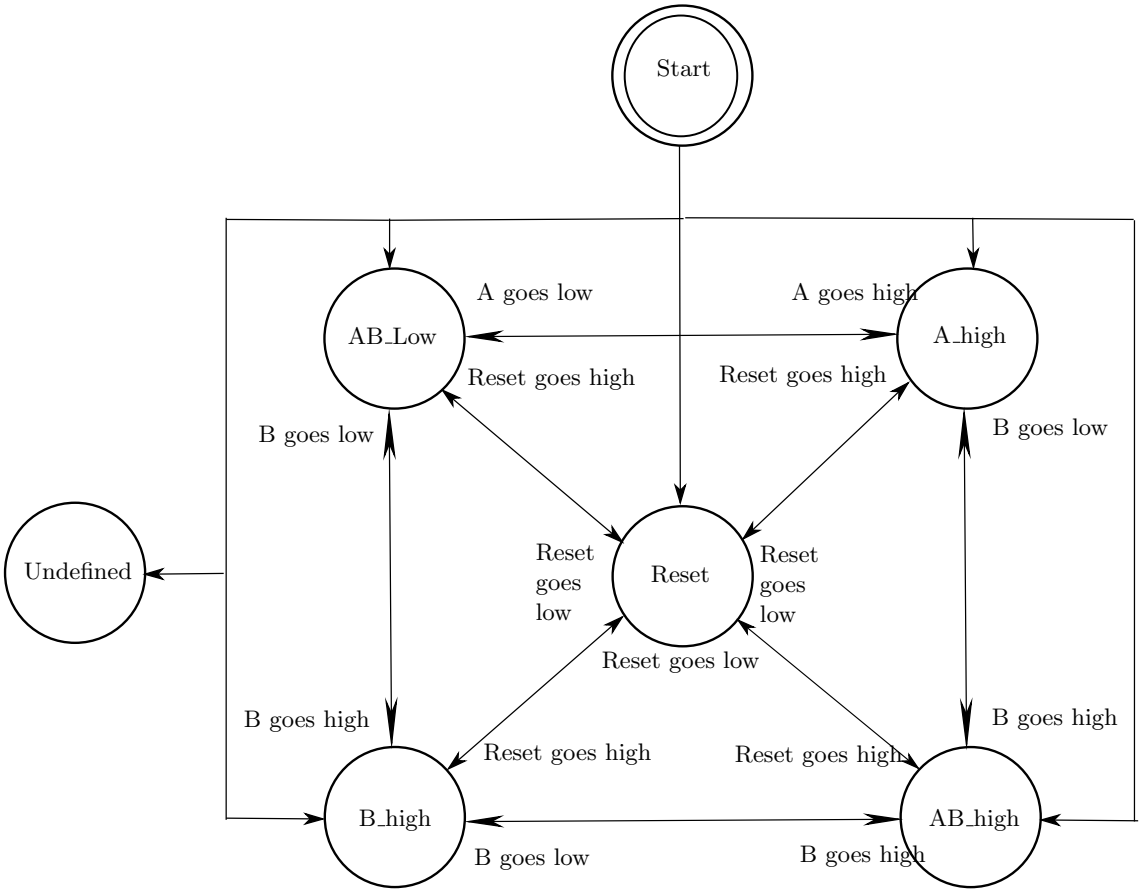

Figure 12

As mentioned above, a reset is needed for the application. This allows for a new home position to be set as desired. The state machine does not make use of a clock singal so the reset is required to be a asyncron reset to avoid adding more complications to the application.

0.4.4 Velocity measure

The decoder is not only responsible for computing the position of the system but also the velocity. The simplest way to estimate the velocity is by diving the change in position with the change in time. The most classic options are either to keep the Δt fixed, measure the position between the interval and then take the difference between x(n) and x(n-1) as shown in equation 0.15. The other is to keep the position fixed and compute how long it takes to reach the fixed Δx like in equation 0.16.

$$v(n) = \frac{x(n) - x(n-1)}{T} \tag{0.15}$$

$$v(n) = \frac{X}{t(n) - t(n-1)} \tag{0.16}$$

The second approach 0.16 is considered the most reliable at slow speeds, which arguably the system does fall under. However, this approach depends on the output to be a "fixed interval pulse train". The system does not fit this criteria, because it generates different frequency waves depending on the supplied PWM value[5].

The first appraoch using equation 0.15 is not hindred by the differnce in frequency it has been chosen.

As the chosen approach requires a fixed time interval one such must be decided. The issue to take into account here is to make sure the interval is neither too short or too long. A too short interval might yield scenario where the velocity seems to be zero even if the system does move. However, on the opposite side, a too long interval could yield a scenario that gives a wrong velocity, if the system changes velocity a lot. If the interval is say 1 second, the velocity at the first $10ms$ be 50 ticks, but it slows down considerably at the $20ms$ mark and hardly moves at $70ms$ mark. This will give an average velocity that might not be as useful and downright misleading as it shows the system to have travelled with a low speed instead of a showing that it was fast at first but slow towards the end. In this case it is important to the system speed up and slow down, so a low interval time of $10ms$ has been chosen.
hej

Forstår hvad d
mener men det
er mudret

[5] https://www.embeddedrelated.com/showarticle/158.php