Slowcontrol for the primary target of the $\overline{\text{P}}$ANDA hypernuclear experiment

von

Nicolas Rausch

Bachelorarbeit in Physik
vorgelegt dem Fachbereich Physik, Mathematik und Informatik (FB 08)
der Johannes Gutenberg-Universität Mainz
am 14. Januar 2016

1. Gutachter: Prof. Dr. Josef Pochodzalla
2. Gutachter: Prof. Dr. Frank Maas

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Mainz, den [Datum] [Unterschrift]

CONTENTS

1 INTRODUCTION

This thesis examines the development of a control system for the primary target of the $\overline{\text{P}}$ANDA[1] hypernuclear experiment. $\overline{\text{P}}$ANDA will be situated at the future nuclear research facility FAIR[2] in Darmstadt and uses an antiproton beam for the creation and high precision γ spectroscopy of double Λ hypernuclei. These investigations provide information about the interaction of Λ hyperons necessary in the progress of a complete understanding of the baryon baryon interaction.

This experiment needs a complex target system. The primary, internal, target is mounted inside a modified beam pipe of the storage ring. During a measurement the primary target has to be moved within the beam. This movement must be very precise. Therefore the target is mounted on piezo motors.

This study focuses on the control of these motors using the EPICS[3] framework which is foreseen for the $\overline{\text{P}}$ANDA slow control system. A prototype of the system is developed and necessary properties for secure measurements are tested.

The first part of this thesis explains the basics of the hypernuclear experiment. Thereby the primary target is focused. In the following sections details on the functionality of the used piezo motors and the CSS[4] frameworks are given. A single board computer for the control of the motors is used. The main part shows the progress done in hardware and software to steer the motors and control them via an EPICS based graphical user interface. Measurements with this setup are presented and finally a conclusion and outlook, including important adjustments on the system are given.

1 Anti-Proton Annihilation Darmstadt
2 Facility for Anti-Proton and Ion Research
3 Experimental Physics and Industrial Control System
4 Control System Studio

2

THE $\overline{\text{P}}$ANDA HYPERNUCLEAR EXPERIMENT

$\overline{\text{P}}$ANDA is one of the major projects of the FAIR facility, an extension to the already existing GSI[1] in Darmstadt. It studies the interactions of antiprotons and fixed target protons or nuclei in a momentum range of 1.5-15GeV/c by using the high energy storage ring HESR [1]. There is a maximum number of 10^{10} antiprotons in the HESR.

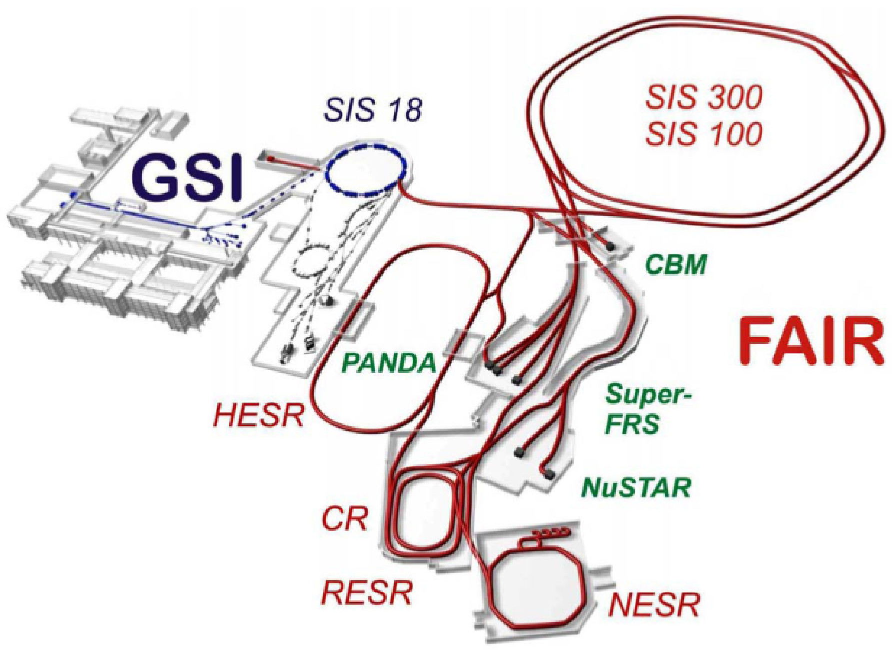


Figure 1: The GSI and the future FAIR facility in Darmstadt.

The $\overline{\text{P}}$ANDA hypernuclear experiment studies the structure of double Λ hypernuclei. A hypernucleus consists of nucleons (protons and neutrons) and at least one hyperon. Hyperons are baryons that consist of at least one strange quark. Since the Λ particle has an up, down and a strange quark it has a strangeness of -1.
The production of double Λ hypernuclei at $\overline{\text{P}}$ANDA proceeds in two steps, see figure 2. First 3 GeV/s antiprotons of the HESR beam interact with protons of the primary, internal, carbon target creating Ξ^- via the reaction:

$$\bar{p} + p \rightarrow \Xi^- + \bar{\Xi}^+ \qquad (1)$$

1 Helmholtzzentrum für Schwerionenforschung

Those particles then leave the primary target and are stopped and captured in absorber layers of the secondary active target. With the reaction

$$\Xi^- + p \rightarrow \Lambda + \Lambda + 28 MeV \tag{2}$$

within a secondary target nucleus a double Λ nucleus is created. This nucleus is in a highly excited state, that deexcites into its ground state by emitting neutrons and photons. If the two Λ decay mesonically it emits two pions, that will be detected in the detector of the secondary target and are used as a trigger for the creation of a double Λ hypernucleus.

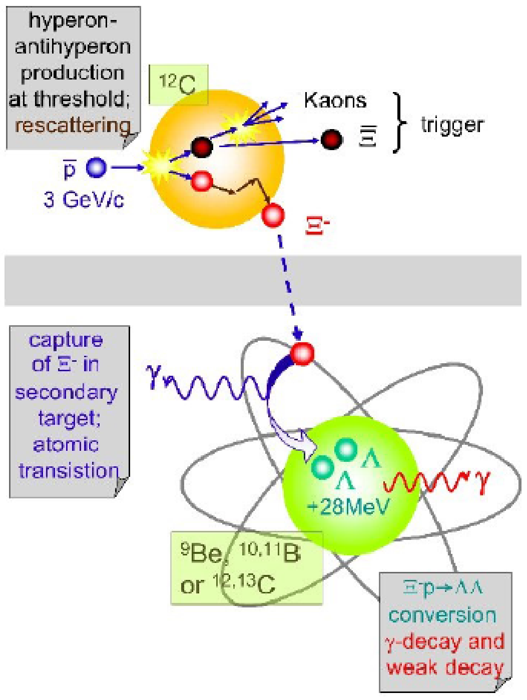

hyperon-antihyperon production at threshold; rescattering
^{12}C
Kaons
Ξ
trigger
$\bar{p}$
3 GeV/c
Ξ^-
capture of Ξ^- in secondary target; atomic transistion
γ
Λ
Λ
+28MeV
γ
^{9}Be, ^{10,11}B or ^{12,13}C
$\Xi^-p \rightarrow \Lambda\Lambda$ conversion γ-decay and weak decay

Figure 2: Illustration of the production process of the hypernuclear experiment. The antiprotons interact with the primary target. The created Ξ^- are captured by a secondary target outside of the beampipe. Hypernuclei are produced in this target. They deexcite by emitting neutrons and photons. These photons are detected and measured using germanium detectors [2].

2.1 EXPERIMENTAL SETUP

The $\overline{\text{P}}$ANDA hypernuclear setup is illustrated in figure 3. It shows the three important parts, primary and secondary target, and the germanium detectors, that are needed for the creation and detection of double Λ hypernuclei.

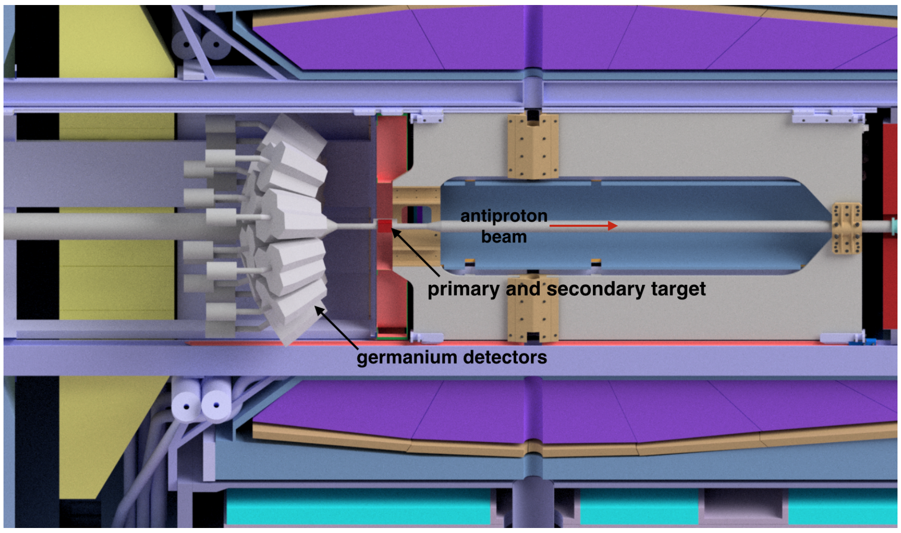
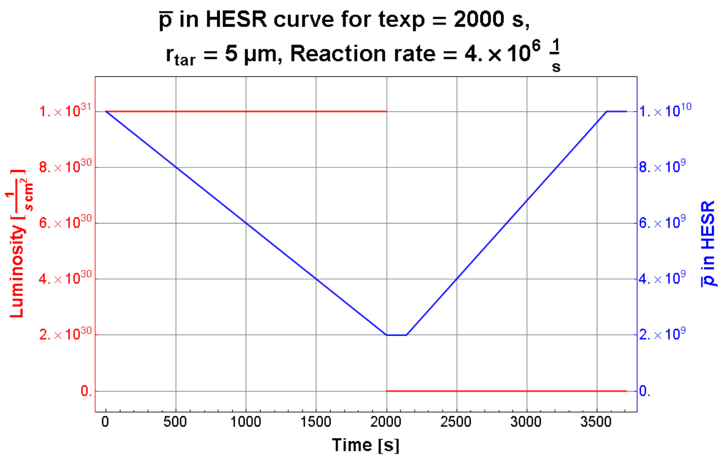
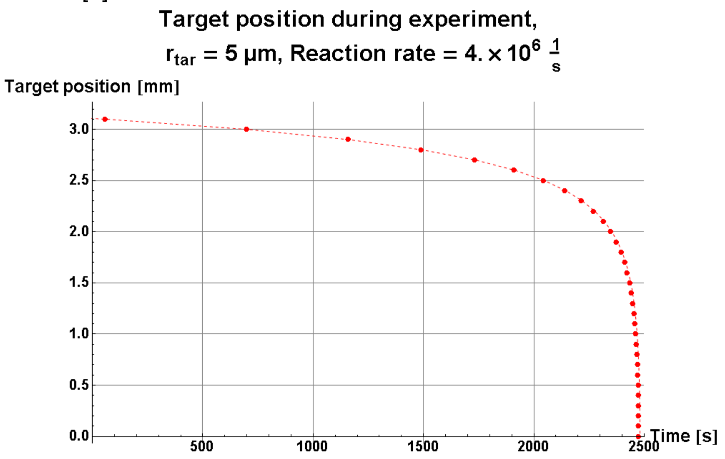


Figure 3: The experimental setup of the $\overline{\text{P}}$ANDA hypernuclear experiment. [2]

The focus of this thesis is the positioning of the primary target in the beam. For that purpose the target is mounted on piezo motors. During a measurement the primary target could be damaged and hence needed to be replaced. Therefore several spare targets are foreseen. The motors holding the primary targets are mounted on a carriage, movable parallel to the beam axis, to enable the replacement of the damaged targets. Todays assumption is that five targets will be mounted on the sled. The piezo motor drives the target through a slit in the absorber material of the secondary target into the beam. This requires that the sled can only be moved when all targets are positioned out of the beam to prevent collisions inside the target vacuum chamber. The control system must ensure this safety behaviour, because otherwise the system could break and the whole experimental setup needed to be opened and repaired. The number of beam particles in a storage ring decreases over time. This is caused by interactions of beam particles with the target nuclei. In the HESR the beam is assumed to have a gaussian shape. For the hypernuclear experiment only a small fraction of the beam may hit the target to achieve the foreseen interaction rate of $4 * 10^6$/s in the measurement time. A movable primary target allows to keep the interaction rate constant by moving the target closer to the center of the gaussian when the total number of antiprotons in the beam decreases. Figure 4a shows one measurement cycle. This includes the measurement time, the time to refill the HESR and the beam preparation time.



(a) Calculation of the luminosity over time and the number of anti-protons during the experiment. [2]



(b) Calculation of the target position during a measure process. The target needs to be moved according to the curve [2].

Looking at the movement plot, the target movement after a measurement time of 2000s increases drastically so that a misplacement leads to a high variation of the interaction rate. Hence it is foreseen to measure for 2000s. In this time the target needs be moved for 0.6mm. To prevent sudden variations in luminosity the target has to be moved as close to the curve as possible. This requires a very precise positioning of the piezo motors. For a better illustration of the sled and motor setup figure 5 gives a detailed illustration on the specific setup.

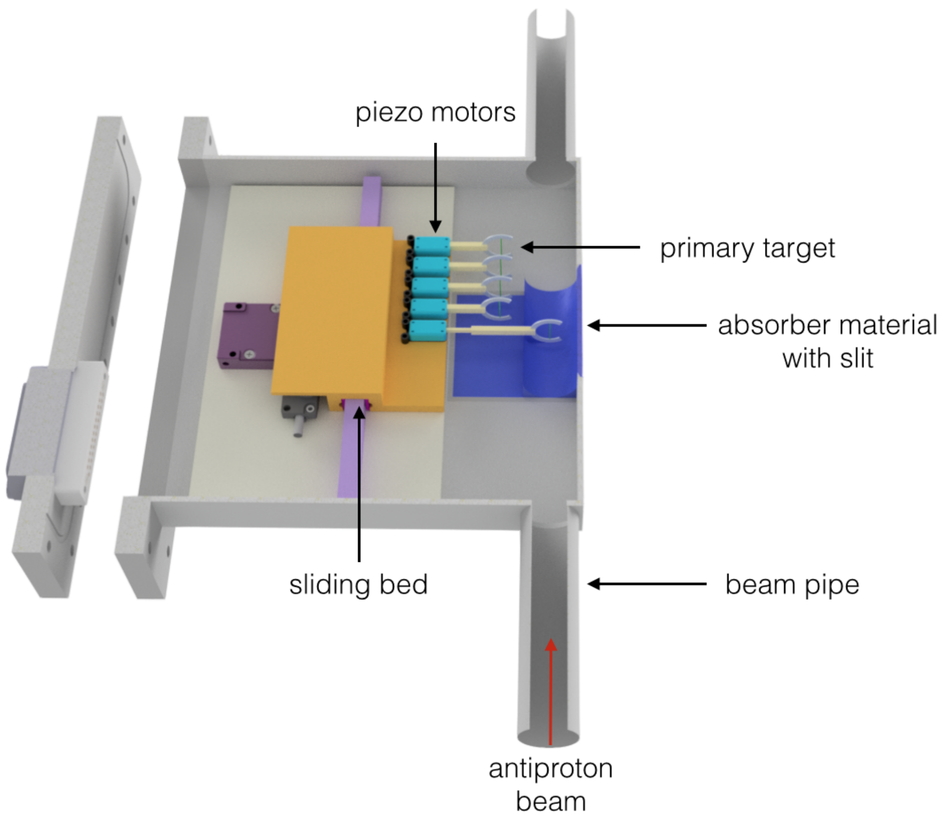


Figure 5: View inside the target chamber. The primary target is placed on the piezo motors, that are mounted on the sled to allow the replacement of damaged targets [3].

3 PIEZO MOTORS

The $\overline{\text{P}}$ANDA hypernuclear experiment requires very small, vacuum and radiation resisting motors than can position the primary target very precisely in the beam. Those requirements are fulfilled by piezo motors and hence used in the setup. For the prototype of the control system the PiezoWave of the company PiezoMotor$^{\text{TM}}$ [4] was used.

Figure 6: The PiezoWave motors with a total length of 14mm, width of 7.2mm and a stroke of 8mm [4].

Even though the PiezoWave are not specified for vacuum conditions they are well suited for setting up a logic for the control system and testing the control of piezo motors in general. The company PiezoMotor$^{\text{TM}}$ also offer motor models that can be used in vacuum applications. In contrast to other step motors, piezo motors dont rely on grease, which makes them useful for vacuum conditions.
In figure 7 a sketch of the functional principle of the PiezoWave is shown.

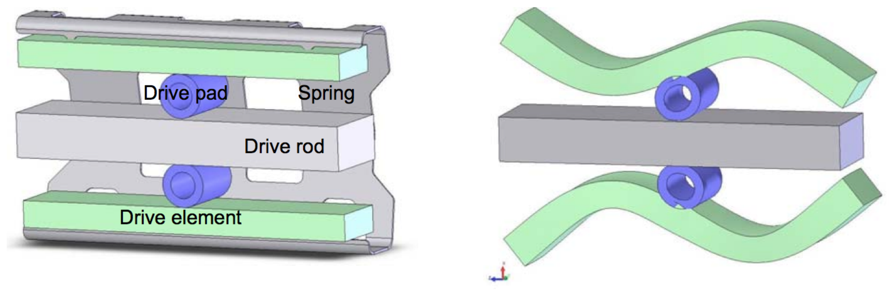


Figure 7: Sketch of the essential parts of the PiezoWave. The left one shows a not activated piezo and the right is electrically activated. The drive element is the piezo itself and vibrates at an ultrasonic frequency when voltage is applied. The drive pad transfers the movement from the piezo elements to the drive rod [4].

The motors offer direct linear drive and have a very low power consumption. When the piezo motor is activated, the piezo elements and the drive pads move, which then causes the drive rod to move 8a. After the first motion cycle 8b, the drive pads have moved as far to the left as possible. In the third stage 8c the drive pads are lifted from the drive rod and enable the piezo elements to reposition the drive pad, so that a new motion cycle is possible 8d.

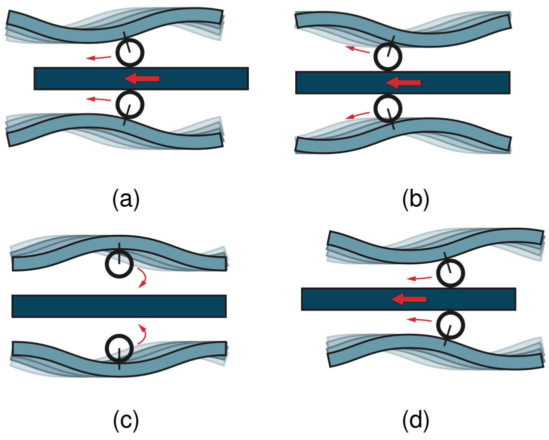


Figure 8: Schematic of the functionality of the piezo motors [4].

The motion is transferred by contact friction from the drive pads to the drive rod. When the drive pads push the drive rod the motor moves. The average step length is about $1\,\mu$m. The drive frequency is up to 100kHz, which makes the motor reach up to 150mm/s at full speed 1. Since the movement of the motor is based on friction, there is no need to keep the motor electrically activated in order to hold its position. When a large force is applied to the drive rod, it will slide without damaging the motor. This makes the piezo motors very durable [5]. Table 1 shows typical motor characteristics.

Table 1: Typical motor characteristics [4]

Speed at no load	150mm/s
Speed at load	50mm/s
Stall force	0.15N
Holding force	0.30N
Stroke	8mm
Step length, average	0.5-$1\mu m$
Life time, cycles 8mm	> 100000

The basic drive electronics of the motor is very simple. Each piezo element consists of two independent parts and can be seen as two capacitors. A motor has two elements and from an electrical point of view they are parallel to each other.
Figure 9 illustrates the electrical interpretation of the motors. The phase shift of A and B determines the movement direction. So in order to change the direction of the movement the phase shift needs to be reversed. Typically two $90°$ phase shifted sinusoidal signals are applied.C1 and C3 are part of the the upper drive element from figure 7 and C2, C4 belong to the lower drive element, see figure 11. One can say that C1 and C2 are left of the drive pad and C3 and C4 are right of the drive pad.

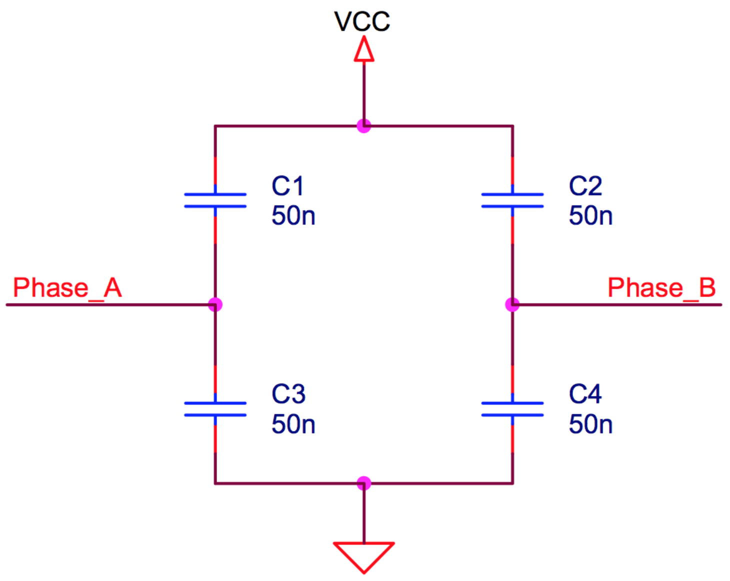


Figure 9: Electrical interpretation of the piezo motors [4].

In 10 typical values for the phase voltages for A and B, as well as for GND and V_{cc} are illustrated. For a better understanding of the functional principle the values of figure 9 are taken and the different voltages applied to the capacitors are explained.

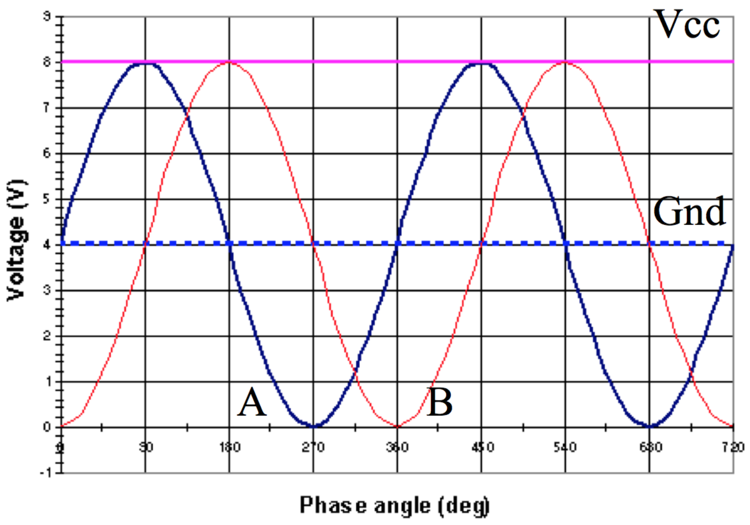


Figure 10: Typical phase voltages for the operating motors [4]. Two sinusoidal signals for phase A and phase B with amplitudes of the control voltage V_{cc}. The peak voltages of the phases should not be more than $2\,V$ higher than V_{cc} and not more than $2\,V$ below GND.

When phase A changes from 4V to 8V the voltage for C1 decreases from 4V to 0V. So the part of the piezo element belonging to C1 returns into its initial position. Meanwhile the voltage on C4 increases from -4V to 0V. The part of the piezo element belonging to C4 changes into its initial position, too, but in opposite direction to C1. Hence the parts of the piezo elements that are directly opposite of each other move in opposite directions.

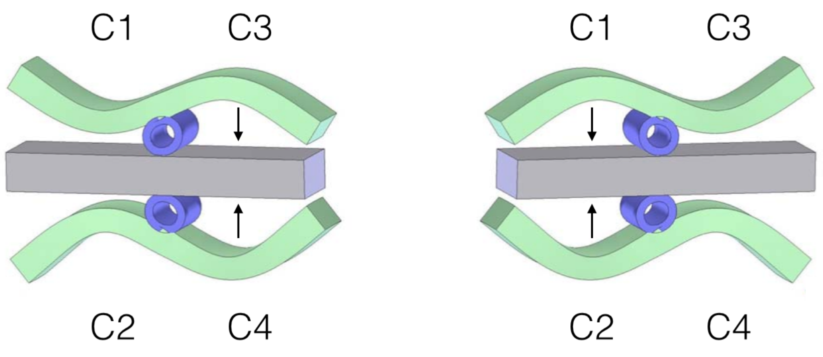


Figure 11: The piezo elements with the movement direction for $+90°$ phase shift (left) and $-90°$ phase shift (right). C1 to C4 show the piezo parts that belong to the capacitors used in the electrical interpretation of the PiezoWave.

As explained above the drive pad nudges the drive rod only for half of the motion cycle. So when there is a phase shift of $+90°$ the right parts of the piezo elements move towards the drive rod and thereby nudge the drive rod to the left and then are lifted from the drive rod. With a phase shift of $-90°$ the left parts of the piezo elements move towards the drive rod and thereby nudge it to the right before they are lifted from the drive rod.

The easiest way to drive the motor is to send a pulse wave over an inductor to the piezo element. This creates an LC circuit and a sine wave signal in the piezo element. In our case the motor is driven via an compact driver PMWD10-01, that is included in the demokit for the PiezoWave motors [6]. The compact driver has six connections, control voltage V_{cc}, phase A voltage and phase B voltage and GND for each of those. Phase A and phase B can be considered as F-signal and R-signal[1], for forward and reverse direction. Applying voltage on the F connector the motor drives forward as long as the signal is applied[2]. If the motor is moved out completely and voltage is still applied to the F connector, the piezo elements will keep operating, but the drive pads no longer nudge the drive rod and instead slide over it. This situation must not happen frequently, because the drive pads can get chafed over time and might consequently lead to a broken motor.

1 In the codes B was used instead of R for Backward.

2 the drive length is only limited by the length of the drive rod

4 CONTROL SYSTEM

4.1 PROTOTYPE SETUP

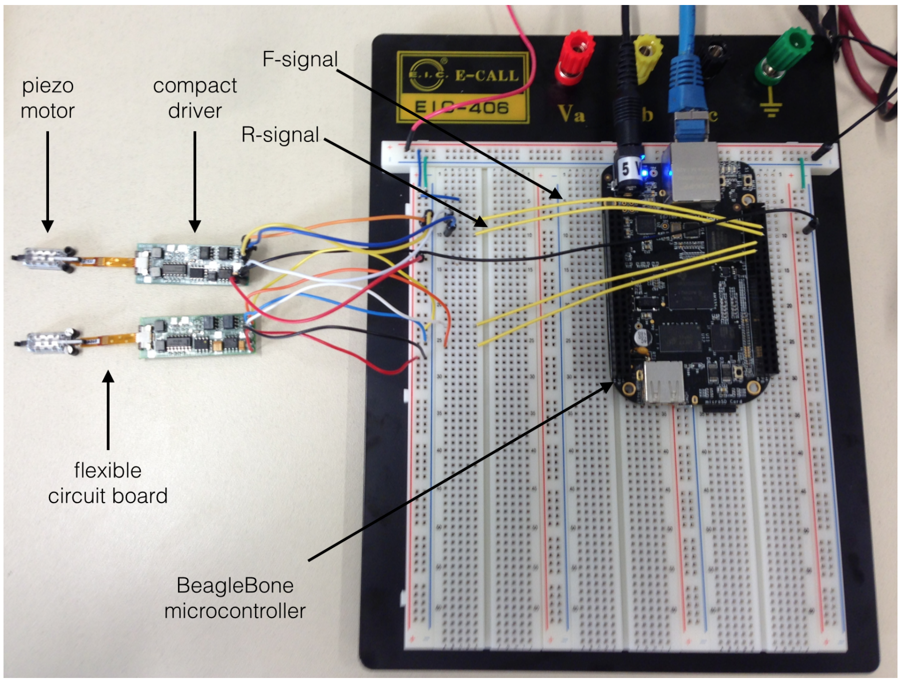


Figure 12: Hardware connection of the piezo motors with a BeagleBone microcontroller via the compact driver.

In figure 12 the prototype setup for the connection of the piezo motors with the BeagleBone Black (see 4.1.1) is given. Two motors are connected using the flexible circuit board and the compact driver (see figure 22 for the schematic in the appendix). The compact driver‘s control voltage V_{cc} is connected to a power supply applying $12\,V$. All GND, including the BeagleBone Black, are connected to the power supplies GND. The F and R pins of the compact drivers are connected to separate GPIO[1] of the BeagleBone for each motor.

1 General-purpose input/output

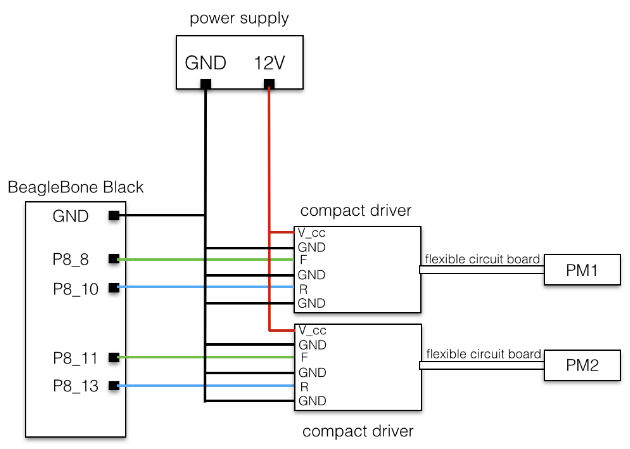


Figure 13: Block diagram for the hardware connection of the prototype.

The GPIO provide $3.3V$, when configured as an output. This allows to drive the motor in the desired direction by setting the digital pin.

4.1.1 *BeagleBone*

The BeagleBone Black is a single-board computer, based on a Texas Instruments ARM Cortex-8 processor[2] running at 1 GHz with 512 MB RAM. It runs a Linux kernel and provides an Ethernet and USB Port. Additionally the boards offers 65 GPIO and serial and I2C connection as well as HDMI [7]. This allows the control of the planned five motors as well as the connection of further data sources. The board can be upgraded with capes, that offer additional features like additional GPIOs, microcontrollers or relays.

Figure 14: The single-board computer BeagleBone Black. [7]

2 Sitara AM3358

4.2 SOFTWARE

The $\overline{\text{P}}$ANDA detector control system is foreseen to use the EPICS[3] framework. Therefore it is used to control the motors via the BeagleBone Black. The IOC[4] for the motors is inplemented on the BeagleBone Black while the graphical user interface is built and running on a desktop PC. Both are connected via Ethernet.

4.2.1 *EPICS*

EPICS is a collection of Open Source software tools, libraries and applications used worldwide in control systems for particle accelerators, telescopes and other large scientific experiments. The installation of the necessary software components of EPICS is done by using precompiled packages[5]. The required packages are:

- the basic EPICS package (epics-dev)
- the asynchronous driver (asyn-dev)
- the driver for stream base communication (stream-dev)
- the driver to control GPIO pins via EPICS (devgpio-dev)
- the add-on for the State Notation Language (seq-dev)

The creation of a new IOC is done by using the included tool makeBaseApp[6]. This creates a basic structure which can be filled. The IOC is started by running the start up script (st.cmd), for more information see the code in the appendix 7.3. [8] In a database (*.db) file all the process variables (PV) are defined as so called records. This database file is newly loaded every time the IOC is started.
The following examples shows how to setup a record in the database file.

```
record( bo, "PANDAHYP:BBB2:PMF1" ) {
  field( DTYP, "devgpio" )
  field( OUT,  "@$P8_8 H " )
  field( ZNAM, "off" )
  field( ONAM, "on" )
}
```

In the first line the type and the PV name of the record is defined, which in this case is an binary output that can be accessed via the PV name *PANDAHYP:BBB2:PMF1*. The DTYP field describes the device type of this record.

3 Experimental Physics and Industrial Control System
4 Input Output Controller
5 http://panda-service.gsi.de/repo/
6 MakeBaseApp creates directories, copies template files into the directories, and expands macros in the files.

In this case a GPIO of the BeagleBone. The OUT field adresses the specific GPIO and whether it is used as active low L or active high H. The other two fields ZNAM and ONAM contain the string that corresponds to the 0 and 1 state. In the given example the record refers to the F-signal of the piezo motor 1 that is connected to GPIO 8 on P8 of the BeagleBone Black. Within the IOC shell the value of this record can be set by using the command:

```
dbpf "<record.field>" "<value>" #database put field
```

For this specific example:

```
dbpf PANDAHYP:BBB2:PMF1 1
```

If no field is given in the PV name the command automatically adresses the value field, e.g. the value of the record itself. To ask for the current value of a record one uses the command:

```
dbgf "<record.field>" "<value>" #database get field
```

This prints the current value of the addressed field. In this case it would return 1 after using *dbpf PANDAHYP:BBB2:PMF1 1*. Setting the recordvalue to 1, sets the GPIO to its high level and the motor drives forward. To drive the motor in reverse direction the value of the record for *PANDAHYP:BBB2:PMF1* needs to be set to 0, because otherwise the movement signal would be applied to both phase inputs of the compact driver not resulting in any movement. After setting the value of the forward direction to 0, the record that corespondents to the reverse direction can be addressed.

```
record( bo, "PANDAHYP:BBB2:PMB1" ) {
  field( DTYP, "devgpio" )
  field( OUT,  "@$P8_10 H " )
  field( ZNAM, "off" )
  field( ONAM, "on" )
}
```

Using

```
dbpf PANDAHYP:BBB2:PMB1 1
```

makes the motor drive in reverse direction. To enable the control of more connected motors one needs to set up a record for each of them. For this it is possible to create templates of records that are used in a substitution file to create multiple similar PVs. The template for such records looks like this:

```
record( bo, "PANDAHYP:BBB2:PM${DIRECTION}${NUMBER}" ) {
  field( DTYP, "devgpio" )
  field( OUT,  "@${PIN} H " )
  field( ZNAM, "off" )
  field( ONAM, "on" )
}
```

This template can be used to create the necessary amount of PVs in the database of the IOC depending on the number of motors to control. The substitution file for the motor records is given below.

```
file "../../db/prepiocNew.db"
{
 pattern{user, DIRECTION,NUMBER,PIN}
{rausch,F,1,P8_8}
{rausch,B,1,P8_10}
{rausch,F,2,P8_11}
{rausch,B,2,P8_13}
}
```

The first line states which database file is loaded. The third line defines the names of the variables that are used in the database file. The following lines are used to set the values for the directions and the pins of the connected motors. Another important record type is the calcoutrecord. This record enables the calculation of data values.

```
record( calcout, "PANDAHYP:BBB2:DELAY1" ) {
  field( SCAN, "1 second")
  field( INPA, "" )
  field( INPB, "1000" )
  field( CALC, "A/B" )
}
```

The SCAN field gives the periodic interval in which a record is processed, which in this example means that every 1 second the value of the record is recalculated. This can be set to shorter times depending on the nature of the signal that has to be calculated. The inputs of the calculation are given in the INP fields. Those can be a value from another record or a constant. The CALC field contains the formula in which the inputs are processed. For example using *dbpf PANDAHYP:BBB2:DELAY1.INPA 1000* changes the value to 1.
EPICS offers many other record types like binary input, analog in and analog out, which are all referenced in the EPICS manual [9]. The binary output and the calcout record are the only ones needed for the control system as it is used right now.
At this point all the commands need to be given by hand and the stopping of the motor has to be set manually by setting the value field of the specific record to 0. EPICS offers a tool to automate and apply a logic to an IOC.

4.2.2 *State Notation Language*

This tool is the State Notation Language (SNL). It allows to programm sequential operations in a real time control system [10]. It is based on the concept of a finite state machine which can be visualized by a state transition diagram. The following figure 15 shows a state diagram for driving a motor in forward direction and declare a step size by setting a delay time after which the GPIO is automatically set back to 0.

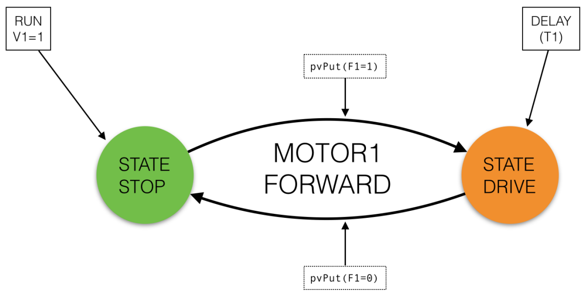
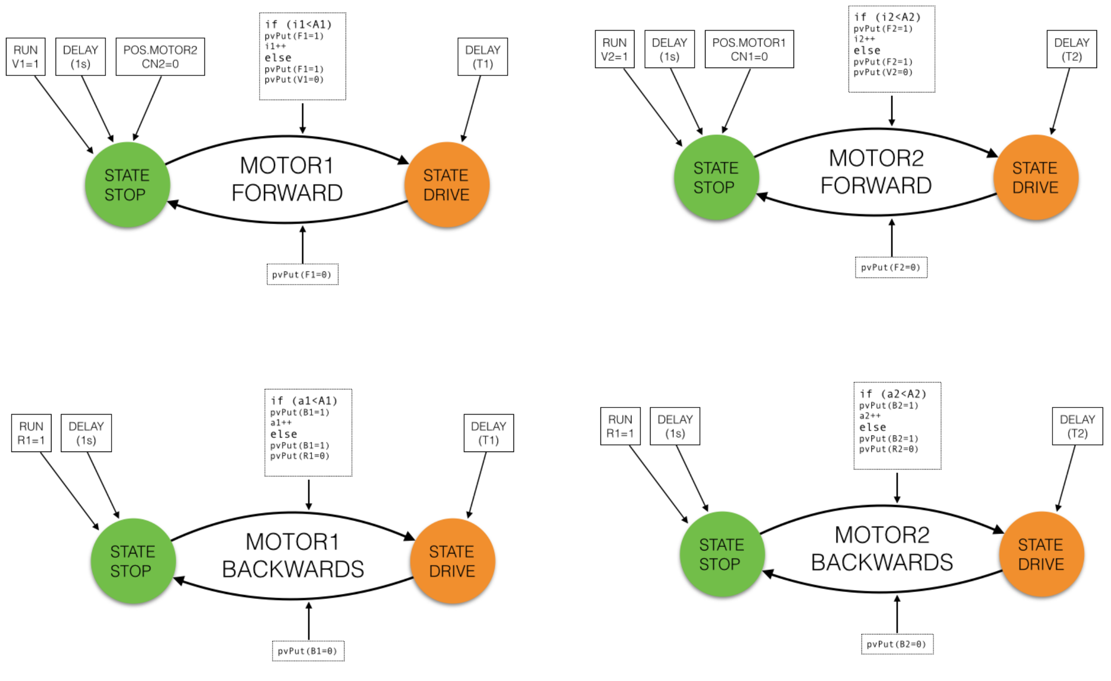


Figure 15: A state transition diagram for the forward movement of motor1. It includes two states. One initial stop state, which belongs to the time the motor is not moving, and the drive state.

The state machine in figure 15 is very simple. By setting the run button to 1 the stop state transitions into the drive state, while setting the GPIO that belongs to the F-signal of the driver board to high. So the initial state is the stop position and by pushing the run button the state transitions. The only condition for the motor to be driven forward is that the value of the record corresponding to the run button is set to 1. The condition to change from drive state to stop state is a delay timer, that can be set via a calcout record. On entering the drive state the delay timer starts running. After that time the commands given within the drive state are processed. In our case the command is to put the GPIO of the F-signal back to 0 and thus stopping the motor. After that the motor is in the initial state again. It is important to remark that the commands are made during a transition from one state to another and not on entering one.
The state diagram that shall be used for the $\overline{\text{P}}$ANDA setup includes important requirements that need to be checked every time before the motors drives forward. There are three conditions for driving the motor forward and two conditions for driving the motor backwards in this motor setup. In order to drive the motor forward, one condition is that every other motor is in initial position, e.g. not driven out at all. The other conditions will be explained when going through the code for driving motor1 forward.
In figure 16 the state diagram for the $\overline{\text{P}}$ANDA control system prototype is given. The example includes only two motors, but the code can be easily extended to five or even more control units.



```
if (i1<A1)
pvPut(F1=1)
i1++
else
pvPut(F1=1)
pvPut(V1=0)
```

```
pvPut(F1=0)
```

```
if (i2<A2)
pvPut(F2=1)
i2++
else
pvPut(F2=1)
pvPut(V2=0)
```

```
pvPut(F2=0)
```

```
if (a1<A1)
pvPut(B1=1)
a1++
else
pvPut(B1=1)
pvPut(R1=0)
```

```
pvPut(B1=0)
```

```
if (a2<A2)
pvPut(B2=1)
a2++
else
pvPut(B2=1)
pvPut(R2=0)
```

```
pvPut(B2=0)
```

Figure 16: The state transition diagram for the $\overline{\text{P}}$ANDA control system, including two motors. For each motor there are two separate state diagrams with two states each. The values going in to the state circles are the conditions that need to return true in order to transition into the next state. During the transition the command showing on the arrow between the states is processed.

The complete state machine for two motors has four separate two state diagrams. For driving the motors forward, there are three conditions going into the stop state and one condition going into the drive state. For backward direction there are only two conditions needed for the stop state, but the rest is the same as in forward direction. The conditions are a button corresponding to a record value that needs to be 1, a wait timer that runs for 1 second every time the stop state is entered and a third condition that belongs to the position of the other motors. In this case it is just the position of motor2. For now the position for each motor is only measured by separate calcout records that count every step the motors make. If a motor makes a step forward it adds $+1$ and if the motor makes a step back it adds -1 to the current calcout value. Only if the value of the calcout records of all other motors is 0 the motor may be driven forward. This condition is not used for driving the motors backwards, first of all because if somehow more motors are driven out at once, it is important to drive them back simultaneously and reset the system. When all those three conditions are fulfilled the motor drives in forward direction until the delay timer of the drive state has finished and sets the value of the GPIO back to 0.

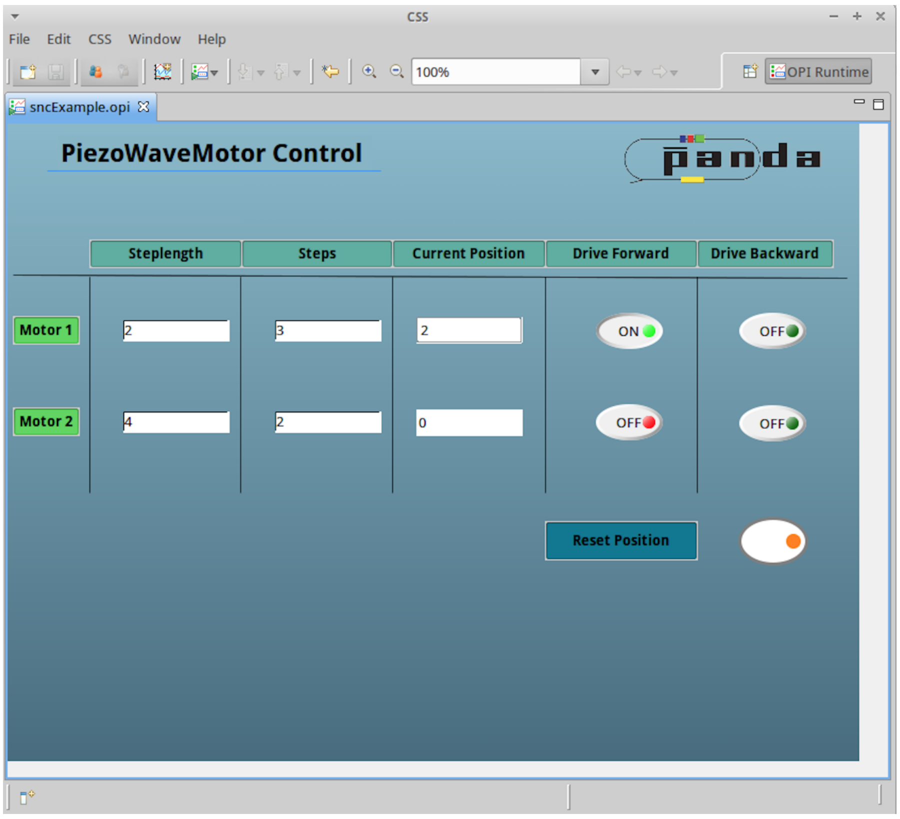
To illustrate how the SNL works, the code for driving the motor1 forward is explained. At the beginning of the program all the variables need to be declared and if used as process variables assigned to specific records.

```
program sncprepioc
int V1;
assign V1 to "PANDAHYP:BBB2:CSSF1";
monitor V1;
int F1;
assign F1 to "PANDAHYP:BBB2:PMF1";
monitor F1;
float T1;
assign T1 to "PANDAHYP:BBB2:DELAY1";
monitor T1;
int C1;
assign C1 to "PANDAHYP:BBB2:COUNTF1";
monitor C1;
float T1;
assign T1 to "PANDAHYP:BBB2:DELAY1";
monitor T1;
int A1;
assign A1 to "PANDAHYP:BBB2:STEPS1";
monitor A1;
int i1=1;
int E1=1;
####MOTOR1FORWARD###
ss F1 {
    state stop {
        when (V1 != 0 && delay(1) && CN2 == 0 ) {
            C1=E1;
            pvPut(C1);
            E1++;
            if(i1 < A1) {
                F1=1;
                pvPut(F1);
                i1++;
                }
            else {
                F1=1;
                pvPut(F1);
                V1 = 0;
                pvPut(V1);
                i1=1;
                }
        } state drive
    }
    state drive {
        when (delay(T1)) {
            F1 = 0;
            pvPut(F1);
        } state stop
    }
}
```

Within a single program one can have several state transition diagrams. Each one is referred to as a state set and a name (line 23). Every state starts with a when statement, in which the conditions are defined under which the state transitions into the next state. The initial state is state stop with three conditions (line 25). The conditions are linked by an AND logic. When all three conditions return true the next bracket is entered. Here the first three commands refer to the calcout record that counts the number of steps (lines $26 - 28$). By using a conditional if-else statement it is possible to let the motor drive a certain number of steps (lines $29 - 40$). A1 refers to a calcout record and its value is the number of steps the motor drives. If the if condition is true the GPIO of the F-signal of motor1 is set to high, by putting F1=1 and the step counter i1 is increased by 1. Then the drive state is entered (line 41). Here the only when condition is a delay timer that can be defined by changing the value of the record that refers to T1 (line 44). When the timer has finished it returns true and the commands of state drive, to set F1=0 (lines $45 - 46$) and thus stop the motor, are processed. This delay time defines how long the signal to move is applied to the motor and thus determines the length of one step, so for shorter delay times shorter steps are made. After the delay has run up and a single step is completed the stop state is reentered (line 47). Again the delay timer of the stop state starts running and the other two conditions are checked. This delay time determines the time between each step and can be change to shorter or longer times depending on the application. The other two conditions are still true and after 1 second the next step is taken. This goes until the else statement is entered. Here the GPIO is set to high, as well, but the value of the run button is put to 0. So when the program enters the stop state, the condition V1 != 0 does no longer return true and the motor remains in its current position.
This example shows how the SNL works and that it is easy to include more restrictions for state transitions. One condition that needs to be integrated in the future is the position of the sliding bed. Furthermore it is important to have an active hardware sensor that measures the position of the piezo motors. For now their position can only be known from the calcout record that counts each step. This counter relies on the software and thus is problematic to be used as the only position reference. If you would change the motor position by hand, the counter would not change, since it basically counts how often the transition from state stop to state drive is made. When changing the motor position by pulling the drive rod out or pushing it in, the software does not recognize this modification.

4.2.3 *Control System Studio*

The graphical user interface for the $\overline{\text{P}}$ANDA detector control system will be built by using the Eclipse based Control System Studio (CSS). It is a collection of tools to monitor and operate large scale control systems [11]. CSS includes a channel access client to communicate to EPICS IOCs and read and access their process variables. This makes it possible to show or change their values. The program offers specific widgets like boolean buttons or text input and text output fields. Those widgets can be linked to process variables and by that make it possible to change the value of a record in our EPICS database from CSS. The graphical user interface for the two motors can be seen in figure 17.



	Steplength	Steps	Current Position	Drive Forward	Drive Backward
Motor 1	2	3	2	ON	OFF
Motor 2	4	2	0	OFF	OFF

Figure 17: Graphical user interface for controlling two piezo motors. Motor2 can not be moved out, because motor1 is not in its initial position. This is highlighted by the red light on the run button for motor2

The GUI includes text input fields for step length and the number of steps, as well as an text output field for the current position of the motors and run buttons for forward and backward direction. In the text input field for step length the delay time of the drive state is declared and in the steps field you define the maximum value of the if condition within the SNL program. When the current position of a motor is not 0 the run button of the other motor for forward direction lights up red to show that it is impossible to drive that one forward. In CSS it is possible to assign rules to your widgets. The current program includes rules that change the color of the light for the run buttons.

When it is green the corresponding motor can be driven forward. When it is red it is not possible to drive the motor forward, because the other motor is already driven out. The rule checks the current position of the motors and works similar to a state condition. The GUI also includes a reset button, that moves every motor to its initial position and sets the value of the current position to 0.

5

PERFORMANCE OF THE CONTROL SYSTEM

The control system is able to steer and position the motors. The $\overline{\text{P}}$ANDA hypernuclear experiment requires fine positioning of the primary target. Over a certain time period the primary target needs to be moved closer to the center of the beam. According to figure 4b the target needs to be moved for 0.6mm in 2000s to achieve the needed average reaction rate. Therefore the performance of the control system is studied by using a USB microscope to measure the length of single steps in both directions for different delay times. The motor is mounted on a metal plate with an attached scale of $1cm$ with $0.5mm$ sections. Then the microscope is adjusted and a picture of the initial position and one after each step is taken. This is done for four steps in both direction. At first the influence of different delay times for the step length are discussed. For each measurement a picture of the initial position and one after four steps is combined into one, whereas the colours of the picture from the initial position are inverted and made semitransparent to see both positions in the superimposed pictures. The difference in the position of the end of the drive rod is measured with the open-source image manipulation program GIMP 2[1], by measuring the difference in pixels. A picture without the drive rod , showing the metal scale, is used for calibration.

1 https://www.gimp.org/

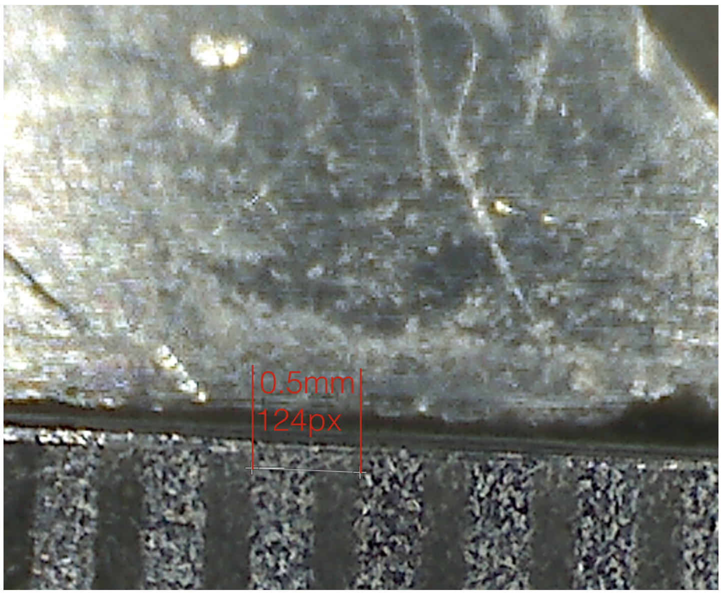
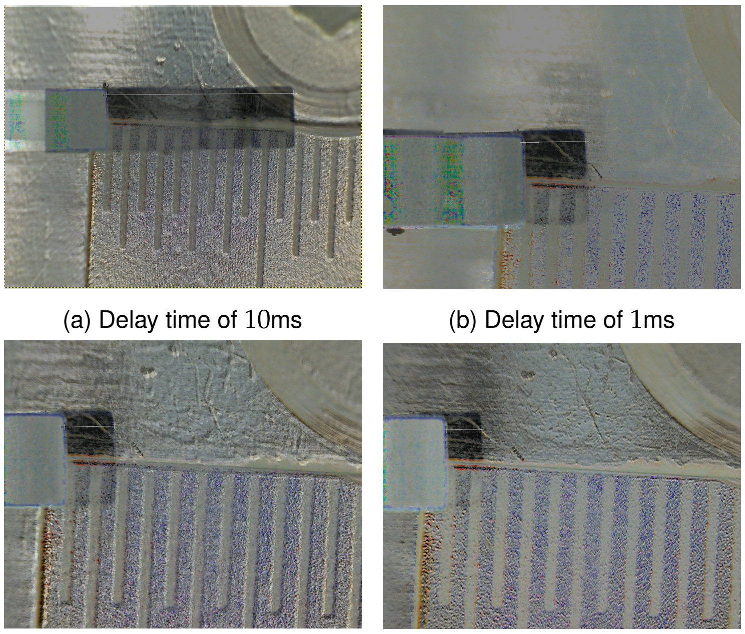


Figure 18: Calibration of the pixel to mm relation. The measured section is 0.5mm and was measured as 124 ± 5 pixels. The error is a result of the resolution of the computer program.

This results to the length for 1 pixel of 0.004 ± 0.00015mm [2]. The following pictures show the measurements for the longest and the smallest delay time. Further pictures can be seen in the appendix.



(a) Delay time of 10ms

(b) Delay time of 1ms

(c) Delay time of $1 \cdot 10^{-1}$ms

(d) Delay time of $1 \cdot 10^{-2}$ms

Figure 19: Overlay of the start and ending positions of the motors. Four steps forward from the initial point for different delay times

2 The error was calculated by dividing 0.5mm by 129px and 0.5mm by 119px. Subtracting the results from each other and dividing it by 2.

For each delay time the length of four steps are measured in pixel and then multiplied with 0.004mm. Dividing the results by four gives us the length of a single step for each delay time. The smallest step length is already achieved at a delay time of $1 \cdot 10^{-2}$ms, which results in 0.17mm for a single step. Measurements were also made for a delay time of 10^{-3}ms, but resulted in the same step length of 0.17mm. Of course the measurements are influenced by errors caused by the resolution of the microscope, the human eye and most importantly the computer resolution when measuring the pixels. To keep those errors small four steps were measured and not only one single step. In the following table 2 the results of the different measurements are presented, including errors. The error for measuring the pixels is estimated to ± 5px, which is ± 0.02mm. For changing from px to mm the error needs to be calculated via gaussian error propagation. p is the length in pixel, m is the length of one pixel in mm and x is the final measured length in mm.

$$x = p \cdot m \qquad with \qquad \Delta x = \sqrt{(m\Delta p)^2 + (p\Delta m)^2} \tag{3}$$

Delay Time	Length of 4 steps	Length of 1 step
10ms	5.33 $\pm 0.20mm$	1.33 $\pm 0.05mm$
1ms	1.18 $\pm 0.05mm$	0.30 $\pm 0.01mm$
0.1ms	0.94 $\pm 0.04mm$	0.23 $\pm 0.01mm$
0.01ms	0.66 $\pm 0.04mm$	0.16 $\pm 0.01mm$
0.001ms	0.68 $\pm 0.04mm$	0.17 $\pm 0.01mm$

Table 2: Length of the motor driven for different delay times.

The limitation in step length is not a property of the piezo motors, that have an average step length of 0.5 to 1 μm [5], but rather a problem of the time the BeagleBone needs to process a command from the SNL program. By using an oscilloscope it was possible to find out that this minimal processing time is 1.2ms. So when the SNL program transitions into the drive state and the delay condition returns true it takes 1.2ms until the GPIO is set back to 0. This was investigated by taking pictures of different delay times with the oscilloscope.

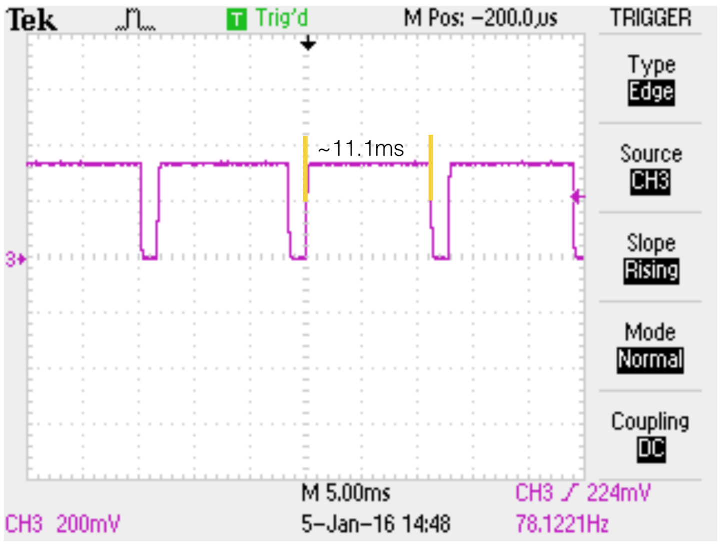


Figure 20: Measured voltage applied to the piezo motor with a delay time of 10ms, with a time scale of 5ms on the oscilloscope. Due to the probe the voltage is $1/10$ of the actual GPIO voltage. Here we have about 330mV, where the actual voltage is 3.3V.

In picture 20 the voltage should be high for 10ms, but the figure clearly shows that there are deviations of about 1.1ms. This deviation explains the non linearity of step length compared to delay time given in table **??**. Where 10ms result in roughly 11.1ms of voltage applied, 1ms results in roughly 2.5ms (see figure 23 in the appendix. So when comparing the step length of 1.33mm to 0.30mm the delay times are rather 11.1ms and 2.5ms. When looking at smaller delay times and smaller scales, the amount of deviation becomes clearer.

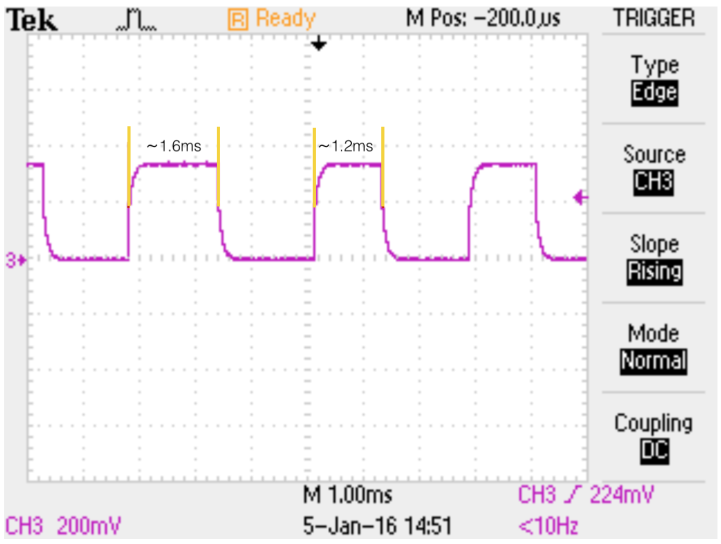


Figure 21: Measured signal applied to the piezo motor with a delay time of $1 \cdot 10^{-3}$ms, with a time scale of 1ms on the oscilloscope. This picture shows jitter, for the same delay time there are variations of 0.4ms. For the voltage scale it is the same as in figure 20.

With a delay time of $1 \cdot 10^{-3}$ms the BeagleBone is not able to switch the GPIO in time and thus causes a limitation in the step length of 0.17mm each. The BeagleBone can not switch the GPIO in less than 1.2ms and therefore it is impossible to achieve shorter step length with this setup.
Another aspect in the oscilloscope recordings is jitter, which is caused because the system used is no real time system and thus the Linux scheduler varies in the order of processing the commands. In figure 21 jitter causes a variation of 0.4ms, which causes fluctuations in step length as well and can not be neglected.
The aim of this study was to look whether or not the step length lies within the range needed for the $\overline{\text{P}}$ANDA hypernuclear experiment. 0.17mm are definitely not accurate enough to achieve precise position of the primary target in the beam. The target needs to be moved 0.6mm in 2000s, which would be 3.5 steps with the current setup. That would lead to big variations in the reaction rate every time the motor makes a step. The deviation of the target position would be to big from the curve in figure 4b. Furthermore jitter causes big variations in step length. Especially when using even smaller delay times, the effect of jitter can become a big problem. Consequently the current setup is not possible to position the target precisely enough to achieve constant luminosity during the measurement.

6

SUMMARY AND OUTLOOK

In this thesis the development of a control system for the positioning of the primary target of the $\overline{\text{P}}$ANDA hypernuclear experiment was examined. Thereby an EPICS based control system was created using a BeagleBone Black. The targets are placed on piezo motors that are connected to the digital outputs of the single board computer via a compact driver board. A graphical user interface was build that enables the control of the piezo motors from any computer that is connected to the same network as the BeagleBone Black. Using a state machine it was possible to implement a control including safety measures for the movement of the motors. These conditions can be defined in a program written in the EPICS implementation of the State Notation Language. This ensures that no motor can be driven forward if any other motor is already inside the beam and hence prevents collisions inside the vacuum chamber. As of yet the conditions are all software based. An active feedback of the motor position is important for the final system to observe mechanically caused misplacements of the targets. That could be a photo sensor that is added on the flexible circuit board, but further investigations of this are necessary to ensure a proper functionality under the conditions of the /Panda hypernuclear experiment.

The major problem of the current setup is the limitation in step length due to the BeagleBone Black. This was studied with an oscilloscope and it was shown that the BeagleBone Black needs roughly 1.2ms to process an incoming command. This is caused by the fact that the operation system installed on the BeagleBone Black is no real time system and therefore the kernel varies in scheduling commands instead of immediately processing the incoming commands the moment they are invoked. This leads to jitter and prevents precise measuring since the step length can vary. A real time system is needed to solve this problem. This could be achieved in two ways. The first one is to implement a real time kernel to the operation system of the BeagleBone Black. These are available and use a special part of the processor, the programmable real-time unit, to process inputs and outputs on the digital interface. This couldn't be done in the thesis because changes to the EPICS implementation of the GPIO control is necessary for this. The second possibility would be to use a microcontroller that is only used for direct control of the motors. The logic would be implemented on the BeagleBone and both would communicate via a serial or I2C connection. This thesis could be used as a basis for this implementation in the future.

7

APPENDIX

Figure 22: Schematic of the compact driver for the piezo motors [6].

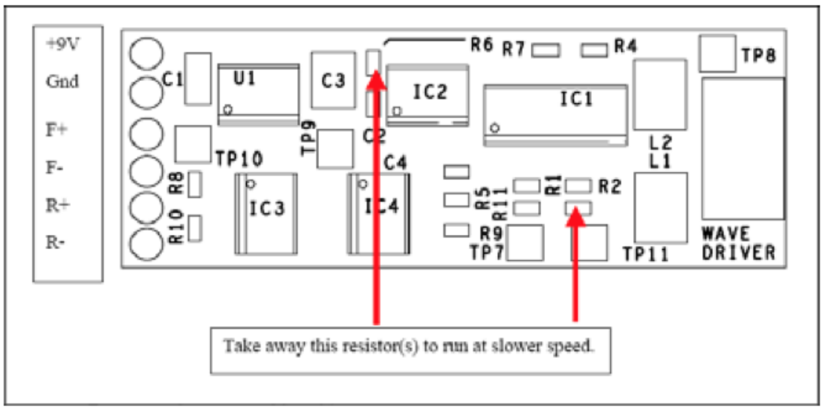
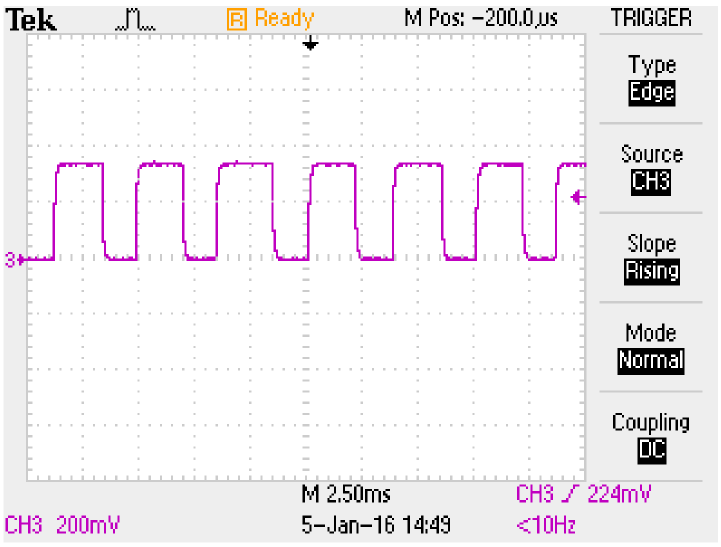


Figure 23: Measurement for a delay time of 1ms.



The following script shows the database file. All records are defined in there. It currently includes the records for the control of two motors and the necessary records for the GUI with CSS.

Code 7.1: The database file

```

##Record substitution file##
record( bo, "PANDAHYP:BBB2:PM${DIRECTION}${NUMBER}" ) {
  field( DTYP, "devgpio" )
  field( OUT,  "@${PIN} H " )
  field( ZNAM, "off" )
  field( ONAM, "on" )
}

          ####Motor1####

###CSS F and B button###
record( bo, "PANDAHYP:BBB2:CSSF1" ) {
  field( DTYP, "devgpio" )
  field( OUT,  "@P8_12 H " )
  field( ZNAM, "on" )
  field( ONAM, "off" )
}
record( bo, "PANDAHYP:BBB2:CSSB1" ) {
  field( DTYP, "devgpio" )
  field( OUT,  "@P8_14 H " )
  field( ZNAM, "on" )
  field( ONAM, "off" )
}

##Delay timer##
record( calcout, "PANDAHYP:BBB2:DELAY1" ) {
  field( SCAN, "1 second")
  field( INPA, "" )
  field( INPB, "1000" )
  field( CALC, "A/B" )
}

##Number of steps##
record( calcout, "PANDAHYP:BBB2:STEPS1" ) {
  field( SCAN, "1 second")
  field( INPA, "" )
  field( CALC, "A" )
}

##Step counter##
record( calcout, "PANDAHYP:BBB2:COUNTF1" ) {
  field( SCAN, "1 second")
}

record( calcout, "PANDAHYP:BBB2:COUNTB1" ) {
  field( SCAN, "1 second")
}

record( calcout, "PANDAHYP:BBB2:COUNTN1" ) {
  field( SCAN, "1 second")
  field( INPA, "PANDAHYP:BBB2:COUNTF1 CA" )
  field( INPB, "PANDAHYP:BBB2:COUNTB1 CA" )
  field( CALC, "A-B" )
}
```

```



          ####Motor2####


###CSS F and B button###
record( bo, "PANDAHYP:BBB2:CSSF2" ) {
  field( DTYP, "devgpio" )
  field( OUT,  "@P8_15 H " )
  field( ZNAM, "on" )
  field( ONAM, "off" )
}

record( bo, "PANDAHYP:BBB2:CSSB2" ) {
  field( DTYP, "devgpio" )
  field( OUT,  "@P8_17 H " )
  field( ZNAM, "on" )
  field( ONAM, "off" )
}

##Delay timer##
record( calcout, "PANDAHYP:BBB2:DELAY2" ) {
  field( SCAN, "1 second")
  field( INPA, "" )
  field( INPB, "1000" )
  field( CALC, "A/B" )
}

##Number of steps##
record( calcout, "PANDAHYP:BBB2:STEPS2" ) {
  field( SCAN, "1 second")
  field( INPA, "" )
  field( CALC, "A" )
}

##Step counter##
record( calcout, "PANDAHYP:BBB2:COUNTF2" ) {
  field( SCAN, "1 second")
}

record( calcout, "PANDAHYP:BBB2:COUNTB2" ) {
  field( SCAN, "1 second")
}


record( calcout, "PANDAHYP:BBB2:COUNTN2" ) {
  field( SCAN, "1 second")
  field( INPA, "PANDAHYP:BBB2:COUNTF2 CA" )
  field( INPB, "PANDAHYP:BBB2:COUNTB2 CA" )
  field( CALC, "A-B" )
}


    ####RESET for CSS####
record( bo, "PANDAHYP:BBB2:RESET" ) {
  field( DTYP, "devgpio" )
  field( OUT,  "@P8_16 H " )
  field( ZNAM, "on" )
  field( ONAM, "off" )
}

record( bo, "PANDAHYP:BBB2:RESET2" ) {
  field( DTYP, "devgpio" )
  field( OUT,  "@P8_18 H " )
  field( ZNAM, "on" )
  field( ONAM, "off" )
}
```

The following script shows the program for the state machine. It includes all states for the motor control and a reset state that can be controlled via CSS.

Code 7.2: Code for SNL state machine

```
program sncprepioc
int F1;
assign F1 to "PANDAHYP:BBB2:PMF1";
monitor F1;
int B1;
assign B1 to "PANDAHYP:BBB2:PMB1";
monitor B1;
int V1;
assign V1 to "PANDAHYP:BBB2:CSSF1";
monitor V1;
int R1;
assign R1 to "PANDAHYP:BBB2:CSSB1";
monitor R1;
float T1;
assign T1 to "PANDAHYP:BBB2:DELAY1";
monitor T1;
int i1=1;
int a1=1;
int A1;
assign A1 to "PANDAHYP:BBB2:STEPS1";
monitor A1;
int C1;
assign C1 to "PANDAHYP:BBB2:COUNTF1";
monitor C1;
int D1;
assign D1 to "PANDAHYP:BBB2:COUNTB1";
monitor D1;
int H1;
assign H1 to "PANDAHYP:BBB2:RESET";
monitor H1;
int E1=1;
int Z1=1;
int F2;
assign F2 to "PANDAHYP:BBB2:PMF2";
monitor F2;
int B2;
assign B2 to "PANDAHYP:BBB2:PMB2";
monitor B2;
int V2;
assign V2 to "PANDAHYP:BBB2:CSSF2";
```

```
monitor V2;
int R2;
assign R2 to "PANDAHYP:BBB2:CSSB2";
monitor R2;
float T2;
assign T2 to "PANDAHYP:BBB2:DELAY2";
monitor T2;
int i2=1;
int a2=1;
int A2;
assign A2 to "PANDAHYP:BBB2:STEPS2";
monitor A2;
int C2;
assign C2 to "PANDAHYP:BBB2:COUNTF2";
monitor C2;
int D2;
assign D2 to "PANDAHYP:BBB2:COUNTB2";
monitor D2;
int H2;
assign H2 to "PANDAHYP:BBB2:RESET2";
monitor H2;
int E2=1;
int Z2=1;
int CN1;
assign CN1 to "PANDAHYP:BBB2:COUNTN1";
monitor CN1;
int CN2;
assign CN2 to "PANDAHYP:BBB2:COUNTN2";
monitor CN2;




ss ssF1 {

  state low {
    when (V1 != 0 && delay(1) && CN2 == 0 ) {
      C1=E1;
      pvPut(C1);
      E1++;
      printf("V1 = 1\n");
      if(i1 < A1) {
        printf("if i < A\n");
        F1=1;
        pvPut(F1);
        i1++;
      }

      else {
        printf("else i=A\n");
        F1=1;
        pvPut(F1);
        V1 = 0;
        pvPut(V1);
        i1=1;
      }
    } state high
  }

  state high {
    when (delay(T1)) {
      printf("Motor 1F Stopped\n");
      F1 = 0;
      pvPut(F1);
    } state low
  }
}


ss ssB1 {

  state low {
    when (R1 != 0 && delay(1)) {
      D1=Z1;
      pvPut(D1);
      Z1++;
      printf("R1 = 1\n");
      if(a1 < A1) {
        printf("if a < A\n");
        B1=1;
        pvPut(B1);
        a1++;
      }

      else {
        printf("else a=A\n");
        B1=1;
        pvPut(B1);
        R1 = 0;
        pvPut(R1);
        a1=1;
      }
    } state high
  }

  state high {
    when (delay(T1)) {
      printf("Motor 1B Stopped\n");
      B1 = 0;
      pvPut(B1);
    } state low
  }
}


ss ssF2 {

  state low {
    when (V2 != 0 && delay(1) && CN1 == 0) {
      C2=E2;
      pvPut(C2);
      E2++;
      printf("V2 = 1\n");
      if(i2 < A2) {
        printf("if i2 < A2\n");
        F2=1;
        pvPut(F2);
        i2++;
      }

      else {
        printf("else i2=A2\n");
        F2=1;
        pvPut(F2);
        V2 = 0;
        pvPut(V2);
        i2=1;
      }
    } state high
  }

  state high {
    when (delay(T2)) {
      printf("Motor 2F Stopped\n");
      F2 = 0;
      pvPut(F2);
    } state low
  }
}

ss ssB2 {

  state low {
    when (R2 != 0 && delay(1)) {
      D2=Z2;
      pvPut(D2);
      Z2++;
      printf("R2 = 1\n");
      if(a2 < A2) {
        printf("if a2 < A2\n");
        B2=1;
        pvPut(B2);
        a2++;
      }

      else {
        printf("else a2=A2\n");
        B2=1;
        pvPut(B2);
        R2 = 0;
        pvPut(R2);
        a2=1;
      }
    } state high
  }

  state high {
    when (delay(T2)) {
      printf("Motor 2B Stopped\n");
      B2 = 0;
      pvPut(B2);
    } state low
  }
}


ss ssR {
  state reset {
    when ( H1 != 0 ) {
      B1 = 1;
      pvPut(B1);
      E1=1;
      Z1=1;
      D1=0;
      pvPut(D1);
      C1=0;
      pvPut(C1);
      B2 = 1;
      pvPut(B2);
      E2=1;
      Z2=1;
      D2=0;
      pvPut(D2);
      C2=0;
      pvPut(C2);
    } state finish
  }
  state finish {
    when (delay(0.5)) {
      B1 = 0;
      pvPut(B1);
      H1 = 0;
      pvPut(H1);
      B2 = 0;
      pvPut(B2);
      H2 = 0;
      pvPut(H2);
    } state reset
  }
}
```

This is the start up script for EPICS. It includes the link to the database that is to be loaded. It is possible to have different databases and use them for different applications.

Code 7.3: Start up script for EPICS

```
#!../../bin/linux-arm/prepioc

## You may have to change prepioc to something else
## everywhere it appears in this file

#< envPaths

## Register all support components
dbLoadDatabase("../../dbd/prepiocNew.dbd",0,0)

prepioc_registerRecordDeviceDriver(pdbbase)
## Configure devgpio driver
GpioConstConfigure( "BEAGLEBONE BLACK" )

## Load record instances
#dbLoadRecords("../../db/prepiocNew.db","user=rausch,DIRECTION
    =F,NUMBER=1,PIN=P8_8")
  dbLoadTemplate("../../db/prepiocNew.sub")
iocInit()

## Start any sequence programs
seq sncprepioc,"user=rausch"
```

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[1] $\bar{\text{P}}$ANDA COLLABORATION: *Technical Progress Report for: $\bar{P}$ANDA*. February 2005

[2] MARCELL STEINEN: *The germanium detector array for the hypernuclear experiment at $\overline{P}$ANDA*, Johannes Gutenberg-Universität Mainz, Diss., 2016. – in preparation

[3] SEBASTIAN BLESER: *The target system for the hypernuclear experiment at $\overline{P}$ANDA*, Johannes Gutenberg-Universität Mainz, Diss., 2016. – in preparation

[4] PIEZOMOTOR: *PiezoWave Linear 0.1 N*, 2015

[5] PIEZOMOTOR: *Datasheets*

[6] PIEZOMOTOR: *PiezoWave Demokit Preliminary Data 20060601*, 2015

[7] BEAGLEBONE BLACK element14: *System Reference Manual for BeagleBone Black*

[8] FELDBAUER, Florian: *Summary of EPICS Workshop and Hands-On Tutorial*. 2015

[9] EPICS REFERENCE MANUAL: *https://wiki-ext.aps.anl.gov/epics/index.php/RRM_3-14*

[10] KOZUBAL, Andy: *State Notation Language and Sequencer Users Guide*. http://www.aps.anl.gov/epics/EpicsDocumentation/AppDevManuals/Sequencer/snl_1.9_man

[11] *Control System Studio*. http://controlsystemstudio.org/

DANKSAGUNG

Zu aller erst möchte ich mich bei Josef Pochodzalla, für seine Vorlesung zur Signalverarbeitung und der Möglichkeit eine Bachelorarbeit zu schreiben, bedanken. Als zweites möchte ich Frank Maas dafür danken, dass er sich als zweiten Gutachter bereit erklärt hat.
Insbesondere danke ich Marcell Steinen für die herausragende Betreuung während meiner Arbeit. Vielen Dank auch an die Arbeitsgruppe von Herrn Pochodzalla und Sebastian Bleser, der mich der Arbeitsgruppe vorgestellt hat.
Daneben danke ich auch Florian Feldbauer für seine Hilfe bezüglich EPICS.
Zu allerletzt danke ich meiner Familie, Freunden und meiner Freundin Lisa für die Unterstützung während des gesamten Studiums.

DEUTSCHE ZUSAMMENFASSUNG

Thema dieser Bachelorarbeit ist die Entwicklung eines Steuerungssystems für das primäre Target des $\overline{\text{P}}$ANDA Hyperkern Experiments.
Die Steuerung basiert auf einer EPICS Umgebung und der zugehörigen State Notation Language (SNL). Das primäre Target wird auf Piezo Motoren montiert und über ein Compact-Board mit einem BeagleBone Black Mikrokontroller verbunden. Für das $\overline{\text{P}}$ANDA Hyperkern Experiment ist es vorgesehen die Steuerung in die ECLIPSE basierende Software Control System Studio (CSS) zu implementieren. Mit Hilfe von CSS wird ein Graphical User Interface erstellt, das es ermöglicht die Motoren zu positionieren und welches über momentane Positionen der Motoren Rückmeldung gibt.
Genauere Untersuchungen des entwickelten Kontrollsystems ergeben, dass das aktuelle System in der Schrittweite der Piezo Motoren limitiert ist. Bedingt durch den BeagleBone Black ist es nicht möglich die Motoren weniger als 0.17mm pro Schritt zu fahren. Für einkommende Befehle benötigt der Mikrocontroller ungefähr 1.2ms um diese zu verarbeiten. Somit ist es nicht möglich Befehlsänderungen in einem Intervall von weniger als 1.2ms ausführen zu lassen. Innerhalb dieser Zeit kann der Motor nicht an und aus geschaltet werden. Damit kann die Länge eines Schritts nicht weiter verringert werden.
Dies erfüllt nicht die Anforderungen des Hyperkern Experiments, da während einer Messung eine konstane Luminostät benötigt wird und daher eine weitaus präzisere Positionierung nötig ist. Mit Hilfe eines separaten Mikrocontrollers ist es möglich eine präzisere Positionierung zu erreichen. Die für die Steuerung nötige Software könnte auf dem BeagleBone Black ablaufen, während der zweite Mikrocontroller über eine serielle Schnittstelle mit diesem verbunden ist und somit Rechenzeit des Kernels eingespart werden kann.