Motor Position Control

System Description

Bonden i Nol
hakanbrolin@hotmail.com
Rev PA1, 2018-01-21, Håkan Brolin

Table of Contents

1 Introduction

1.1 Purpose

This project investigates how to control the position of a DC motor shaft using PID controller.

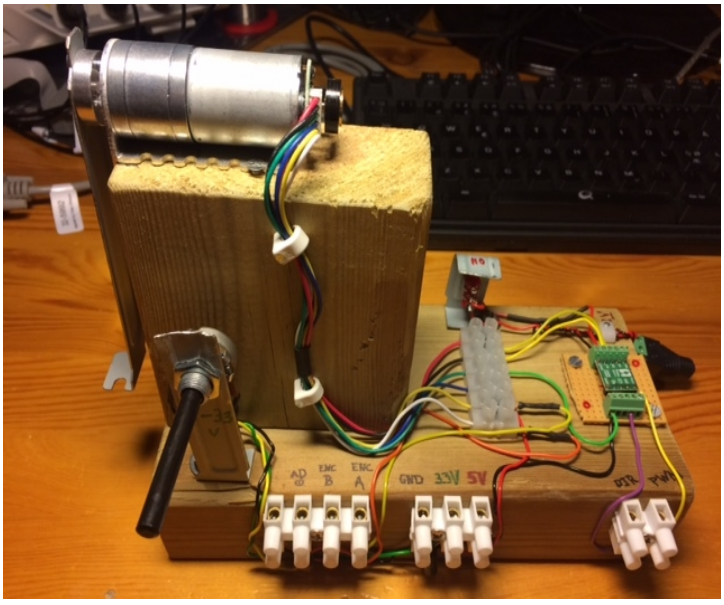
1.2 Resources

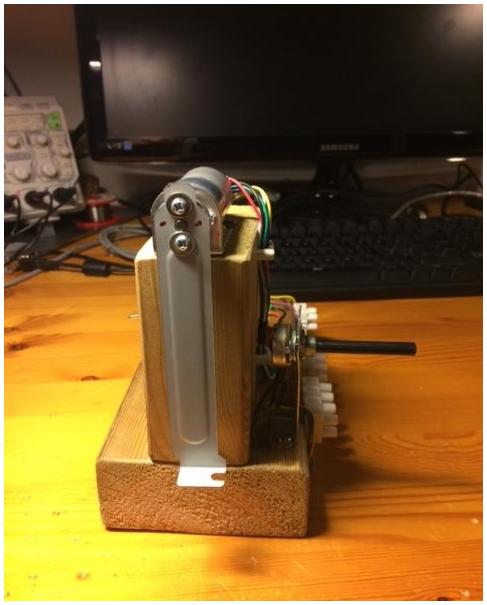
All resources for this project, including source code and documentation can be found at:

https://github.com/emwhbr/dcmotor

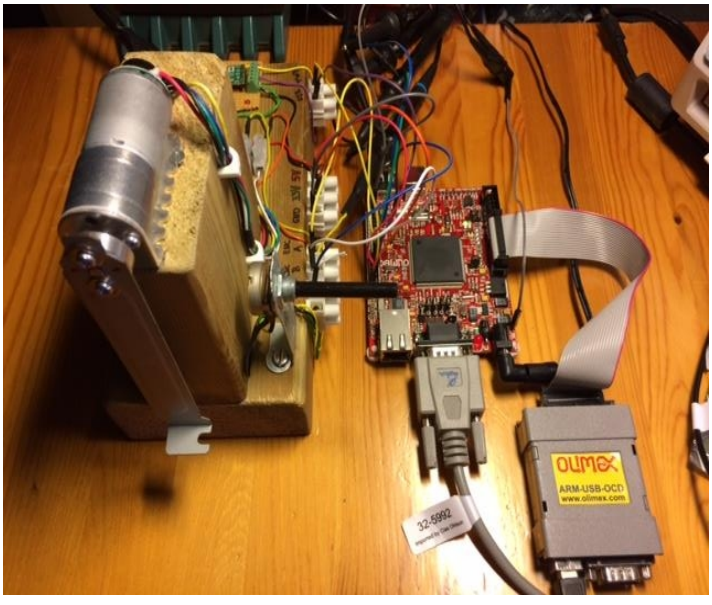
1.3 Test system

DC motor with bracket mounted on shaft and potentiometer for commanded position.





DC motor connected to controller development board. RS232 connector and JTAG adapter.



1.4 Controller platform

Controller application is implemented in C and executes on a ARM9 32bit micro-controller (MCU).

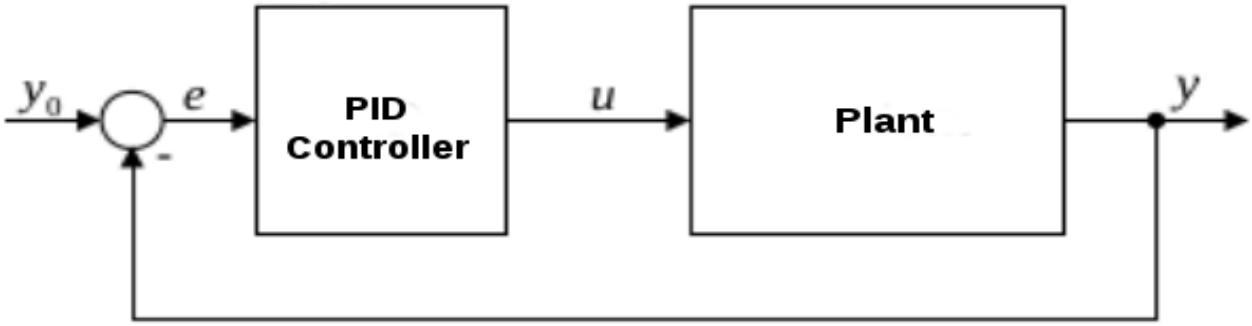
A “bare metal” application approach was selected to get low latency and overhead, and to achieve high real-time performance. This decision made the application more complex then when using an operating system (Linux). The learning curve is steeper and the lack of support of system resources makes it harder to get up and running in a small amount of time.

But the “bare metal” approach makes it possible to customize the application with a minimum use of system resources. It's also an interesting way to get to know the hardware and increases your knowledge about various aspects of the system, like start-up sequence, interrupt handling and more.

2 PID controller

2.1 Theory

A closed-loop embedded PID control system:



The PID controller function is described as:

$$\mathrm{u}(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt}$$

The discrete form of the algorithm is:

$$u(n) = K_p e(n) + K_i \sum_{k=0}^{n} e(k) + K_d \left(e(n) - e(n-1)\right) \qquad K_i = \frac{K_p T}{T_i} \qquad K_d = \frac{K_p T_d}{T}$$

The derivate on error *e(t)* is identical to the negative of the derivate on process variable *y(t)*, except when set point changes. When set point changes, derivate on *e(t)* results in an undesirable action on controller output which is called the “derivate kick”. Basing the derivate term on the process variable only results in an improved PID controller.

This controller lacks the impact of the “derivate kick” when the set point changes:

$$u(n) = K_p e(n) + K_i \sum_{k=0}^{n} e(k) - K_d \left(y(n) - y(n-1)\right)$$

This formula is used in the motor position controller software.

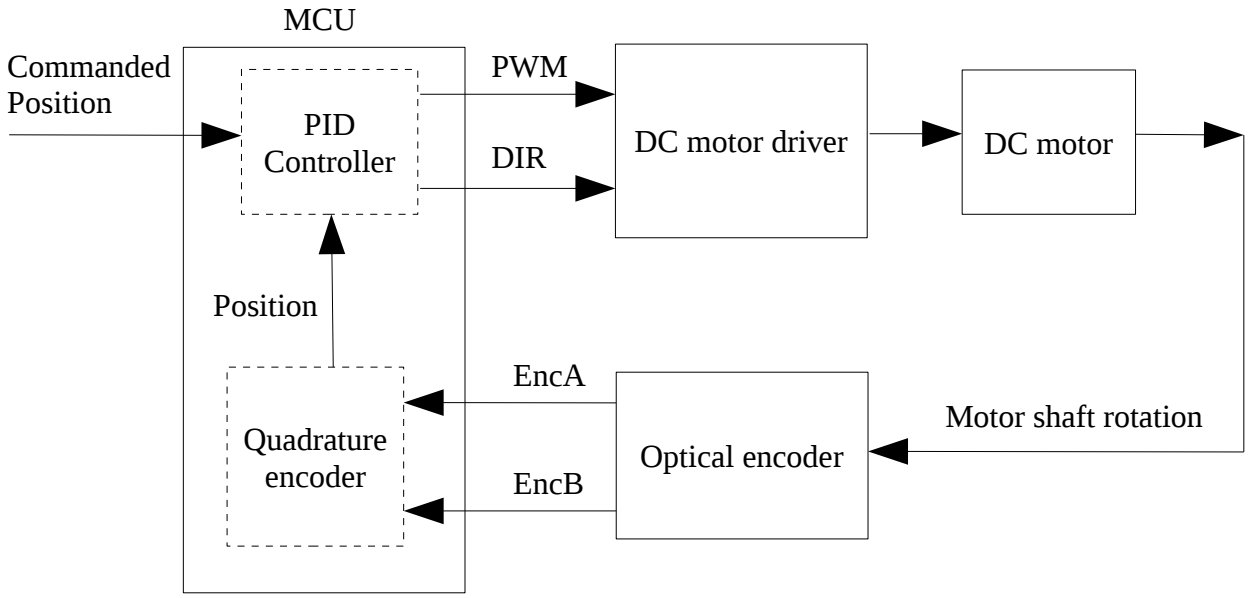
2.2 References

References on PID controller theory and how to apply them in an embedded system:

- PID control and derivate on measurements.
  http://controlguru.com/pid-control-and-derivative-on-measurement
- PID without a PhD, Tim Wescott.
  http://www.wescottdesign.com/articles/pid/pidWithoutAPhd.pdf
  http://m.eet.com/media/1112634/f-wescot.pdf

2.3 Motor position controller

The motor position controller system is implemented according to figure below:



The DC motor driver hardware controls the direction and speed of the DC motor.

The output from the PID controller software is a PWM duty cycle value ranging from -65000 .. 65000. The absolute value of this output is feed as a PWM signal to the motor driver hardware. Positive values results in a clockwise direction signal and negative values results in an anti-clockwise signal being generated.

Motor shaft rotation is transformed by the optical encoder hardware into two output channels that indicates both position and direction of the rotation. The quadrature encoder software transforms these signals back to an absolute position. This is an interrupt driven process where both encoder outputs generates an interrupt to the MCU.

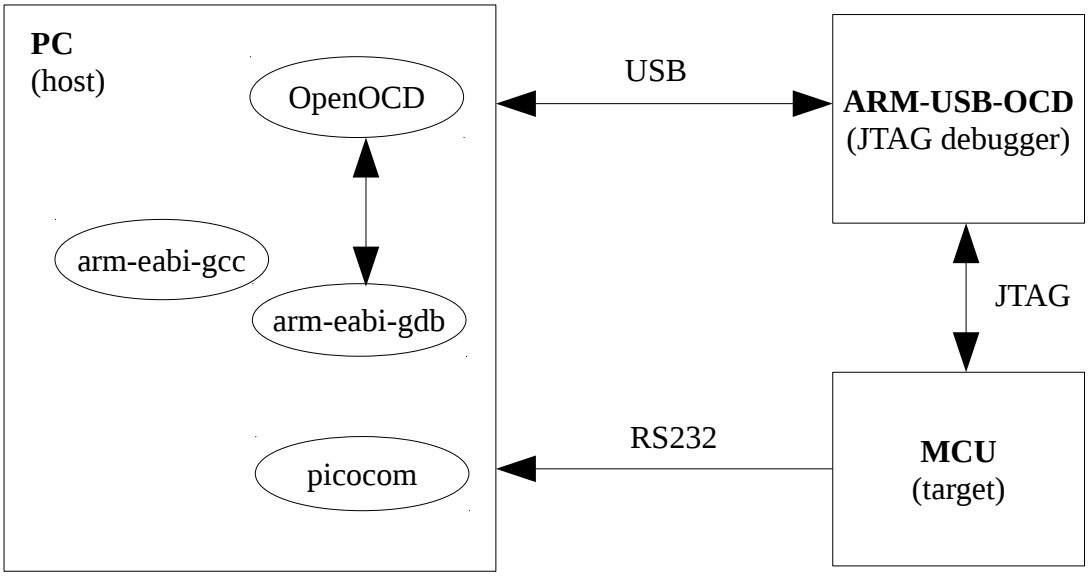
The PID controller software executes at a constant frequency by a timer interrupt. This means that the sampling of the actual position is done at a consistent interval with a minimum of variations.

3 Development tools

3.1 Development system

The development host is a PC running Linux Mint 17 (64-bit).

The host software includes a cross-toolchain and other utilities for target system development.



3.2 Toolchain

Linaro GCC bare-metal ARM cross-toolchain (6.3.1, 2017-05) with the following features:

- 32-bit
- little-endian
- soft-float

This version was downloaded from:

```
https://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/arm-eabi/gcc-linaro-6.3.1-2017.05-x86_64_arm-eabi.tar.xz
```

Other versions are available at:

```
https://releases.linaro.org/components/toolchain/binaries/
```

3.3 OpenOCD

Debugging and programming of the target is performed via JTAG, using the ARM-USB-OCD (Olimex) as interface between OpenOCD and the target system.

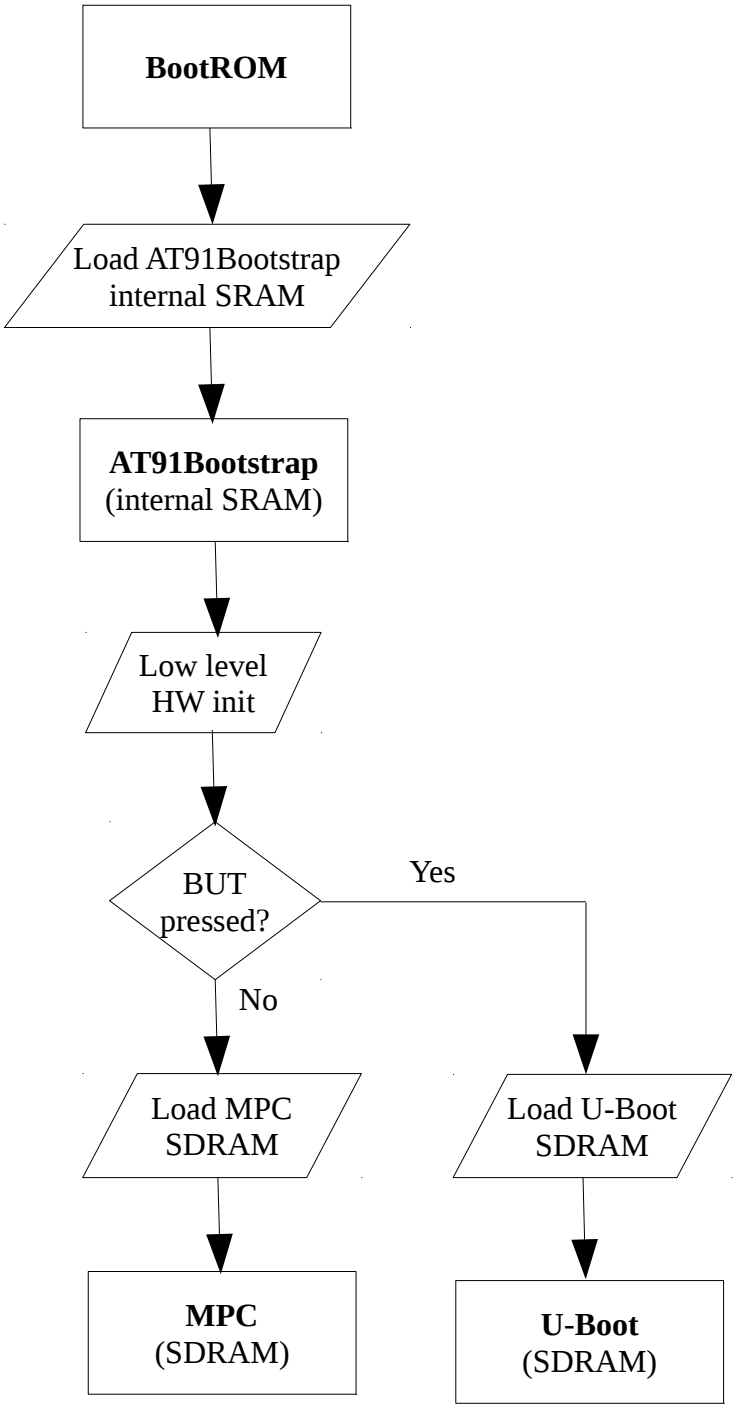
Further information can be found at:

https://www.olimex.com/Products/ARM/JTAG/ARM-USB-OCD/

4 Software

4.1 Startup

The boot sequence of the system software is shown in the figure below.



The different software components are described in the following sections.

4.2 BootROM

BootROM is resident in the MCU AT91SAM9260 (stored in ROM). This “first stage” boot loader can launch a “secondary stage” loader from different locations (SPI Data Flash, NAND Flash or USB).

The source code for this component is not included in the project.

4.3 AT91Bootstrap

AT91Bootstrap is the “secondary stage” loader, stored in NAND Flash. The code is downloaded into the internal SRAM by *BootROM*, followed by a jump to the first address of the SRAM . The binary image size must be less than 4K, corresponding to the SRAM size for the AT91SAM9260 . The code needs to be kept small and only performs basic hardware initialization tasks. This includes initialize peripheral I/O, enabling I-cache, setup clock speed and configuration of the SDRAM controller.

AT91Bootstrap initializes the clock speeds according to:

Master clock: 102.4 MHz
CPU clock: 204.8 MHz

The source code for this component is included in the project.

4.4 U-Boot

U-Boot is the “third stage” loader, stored in NAND Flash. The code is downloaded into SDRAM by *AT91Bootstrap*, followed by a jump to the start address (0x23F0_0000). *U-Boot* continues the more complex hardware initialization of the board (e.g. Ethernet, USB, SD/MMC). It includes support for several commands and different communication protocols.

The source code for this component is not included in the project (binary image is included).

4.5 MPC

MPC is the actual “bare metal” application, stored in NAND Flash. The code is downloaded into SDRAM by *AT91Bootstrap*, followed by a jump to the start address (0x2220_0000).

MPC (Motor Position Control) performs the positioning of the DC motor shaft. It includes the PID controller algorithm and other software functions that are needed for the project objectives.

The source code for this component is included in the project.

5 Development board

5.1 Board

SAM9-L9260 development board (Olimex). Further information can be found at:

https://www.olimex.com/Products/ARM/Atmel/SAM9-L9260

5.2 MCU

MCU: AT91SAM9260 (Atmel), 32 bit
Core: ARM9 core family (ARM926EJ-S)
Architecture: ARMv5TE

5.3 Jumpers

Jumpers on the board needs to be configured according to:

BMS_LOW State: **Open**
Forces the BMS-pin (Boot Mode Select) of the CPU to high state.
This selects embedded ROM as boot device after reset, thus activating BootROM.

DF_E State: **Open**
Forces the CS-pin of the SPI Data Flash to high state so that the chip can't be used.
When no Data Flash is found, BootROM will look for the “secondary stage” loader in the NAND Flash instead.

NANDF_E State: **Closed**
Allows the CE-pin of the NAND Flash memory to be controlled by software.
If this jumper is open, the chip can't be enabled or used.

5.4 NAND Flash

The NAND Flash memory is organized according to the table below:

Software component	Address range (hex)	Size
AT91Bootstrap	0x0000_0000 - 0x0000_0fff	4 KB
Spare	0x0000_1000 - 0x0001_ffff	124 KB
MPC	0x0002_0000 - 0x0003_ffff	128 KB
Spare	0x0004_0000 - 0x1cef_ffff	462,75 MB
U-Boot	0x1cf0_0000 - 0x1cfd_ffff	896 KB
U-Boot environment	0x1cfe_0000 - 0x1cff_ffff	128 KB
Spare	0x1d00_0000 - 0x1fff_ffff	48 MB

The memory is 512 MB, with “block” size of 128 KB (0x20000) and “page” size of 2 KB (0x800).

5.5 External connector EXT

Pin	MCU pin	MCU function	Application
1, 2	-	-	3.3V
4	-	-	5V
39, 40	-	-	GND
5	PB0	PIO Controller B Peripheral B TC3, TIOA3	PWM duty output control signal. Frequency=781.3Hz (T=1.28ms) Connected to MAX14780 speed control input.
7	PB1	PIO Controller B PIO output	Motor direction control signal. High: clockwise, Low: counterclockwise Connected to MAX14780 direction input.
9	PB2	PIO Controller B PIO output	Debug pin#1. Indicates DC motor shaft position error +- 1.5%.
11	PB3	PIO Controller B PIO output	Debug pin#2. Indicates stop state entered due to a request to change motor direction.
13		PIO Controller B PIO input	Encoder A input. Interrupt enabled on pin.
15		PIO Controller B PIO input	Encoder B input. Interrupt enabled on pin.

5.6 External connector AEXT

Pin	MCU pin	MCU function	Application
1	GNDANA	Analog ground	GND
2	-	-	-
3	PC0	PIO Controller C Peripheral A AD0	Analog input channel (10-bit, ADC=0..1023). Connected to external potentiometer. Determines PID set point for motor shaft during calibration.
4	-	-	-
5	ADVREF	Analog reference	3.3V
6	VDDANA	Analog power supply	3.3V

6 DC motor components

6.1 DC motor

Brushed DC (12V) motor with gearbox and quadrature encoder: Pololu, part#3240.

https://www.pololu.com/product/3240

Specifications:

Free-run current:	200 mA (stall current: 2100 mA)
Free-run speed:	220 rpm (3.67 rev/s, 273 ms/rev)
Gear ratio:	34.014:1
Encoder:	48 CPR, quadrature (34.014 x 48 = 1632,672 counts per revolution)

6.2 DC motor driver

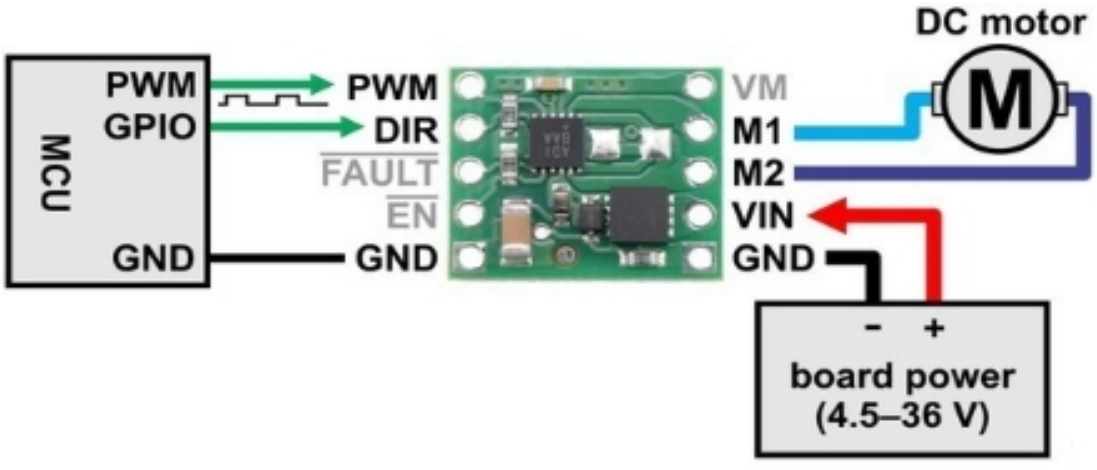
Motor driver carrier board for MAX14870: Pololu, part#2961.

https://www.pololu.com/product/2961

Specifications:

Motor supply voltage:	4.5 – 36 V
Output current:	1.7 A (2.5 A peak)
PWM frequency:	50 kHz (max)

Application circuit:



7 MPC

7.1 Software layout

The “bare metal” software is organized in the following directory structure:

```
sw/motor_position_control
          |
          |-------- bsp
          |
          |-------- core
          |
          |-------- proc
```

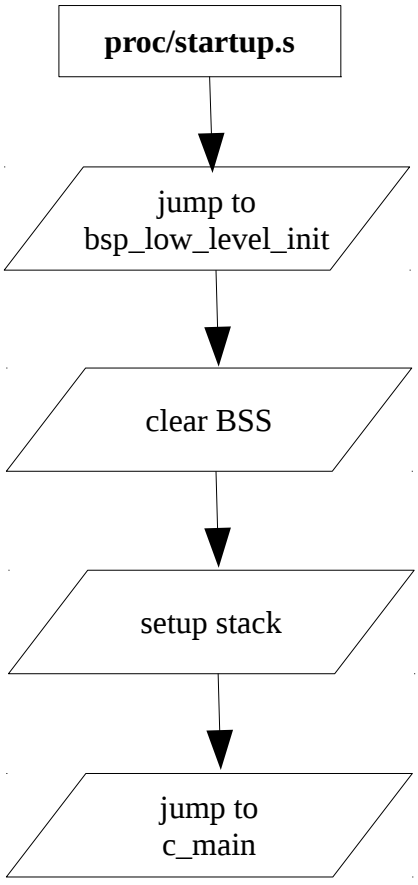
Directory *bsp* includes board support functionality.

Directory *core* includes application specific functionality.

Directory *proc* includes processor specific functionality.

7.2 Startup

The startup sequence for the “bare metal” application:



Link address: 0x2220_0000 (SDRAM)
Entry point: _reset

Implemented in C (`bsp/bsp.c`)
Setup ARM exception vectors
Remap SRAM to address 0x0000_0000

Unitialized data set to zero

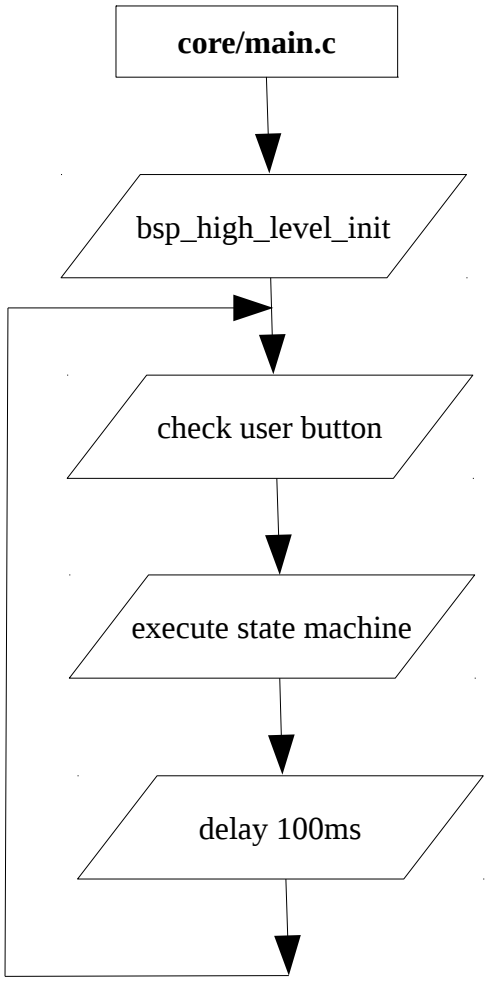
Fill stack with known pattern
Initialize stack pointers

Implemented in C (`core/main.c`)

The application layout is controlled by the linker script `core/mpc.ld`.

7.3 Flowchart

The application main flow is depicted in the flowchart below:



Entry point: c_main

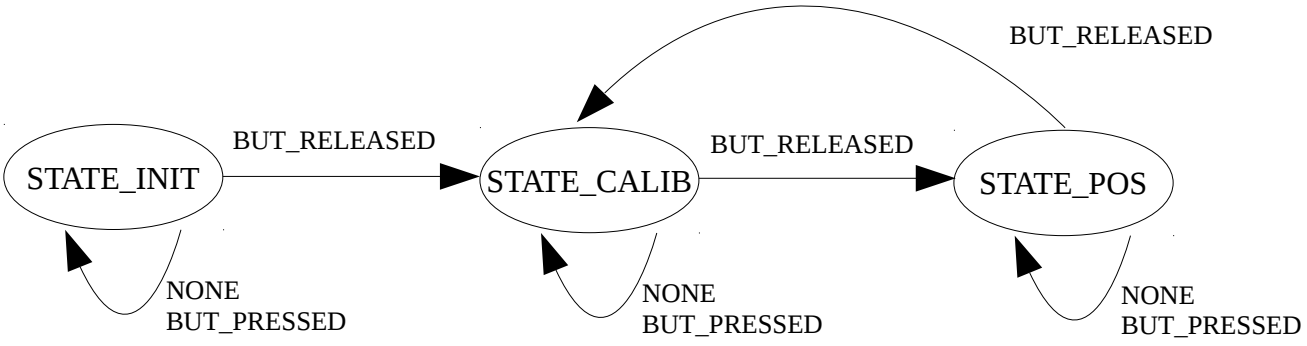
Implemented in (`bsp/bsp.c`)
Initialize board hardware components
Setup timer interrupts
Setup interrupt controller (AIC)

Button value controls the state machine

Implemented in (`core/mpc_fsm.c`)

7.4 State machine

The value of the user button controls the state machine:



`mpc_core_initialize`

Disable PID controller

`mpc_core_calibrate`

Disable PID controller
Update PID set point
BUT_PRESSED ==> zero shaft

`mpc_core_position`

Enable PID controller

The actual update of the PID controller is executed by a timer interrupt at constant frequency.

8 PID controller test#1

8.1 Conditions

- PID controller frequency: 100Hz (T=10ms)
- PID parameters:

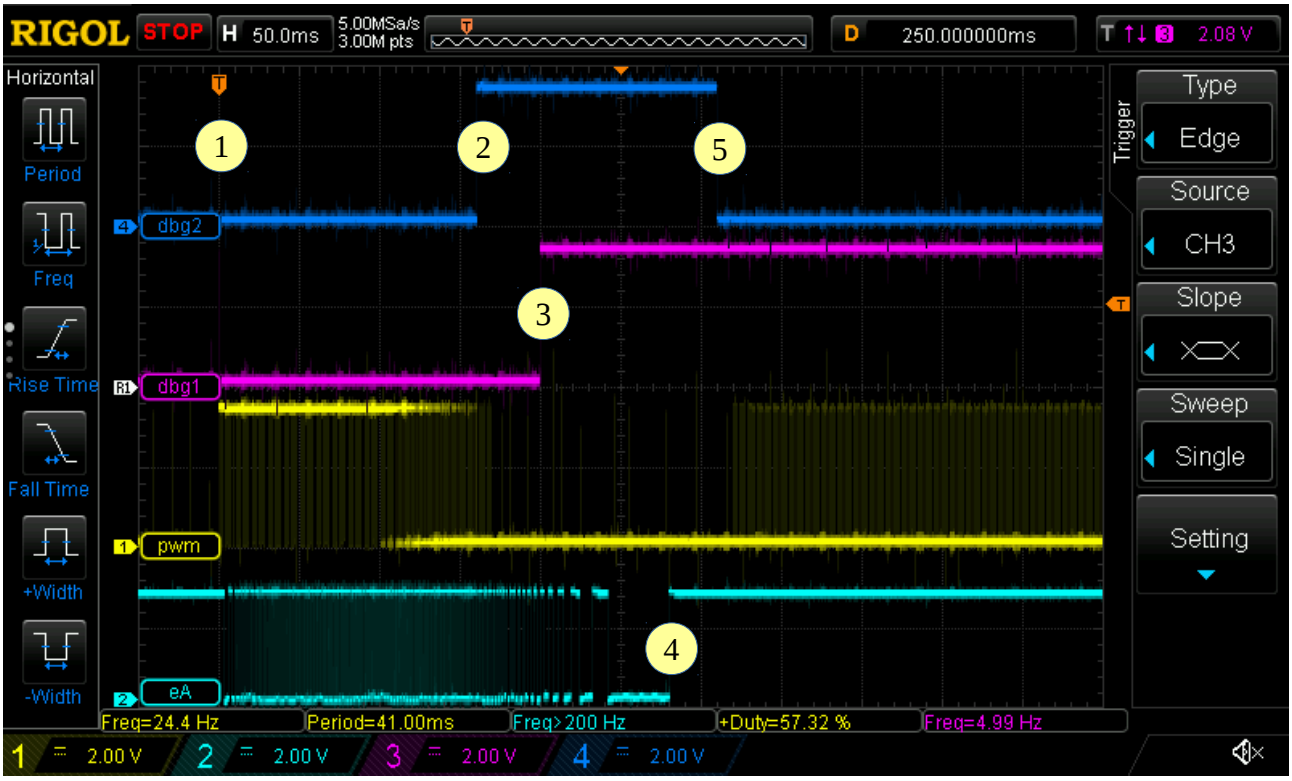
  P=250
  I=0
  D=700

- Integral part (I) was set to zero. Several experiments showed that adding this component would either slow down the system, or make it unstable. The typical I-values that were tested were unfortunately not noted.

8.2 PID controller test#1.1

Conditions:

- Motor shaft position set point: 0x330 = 816 (180°)



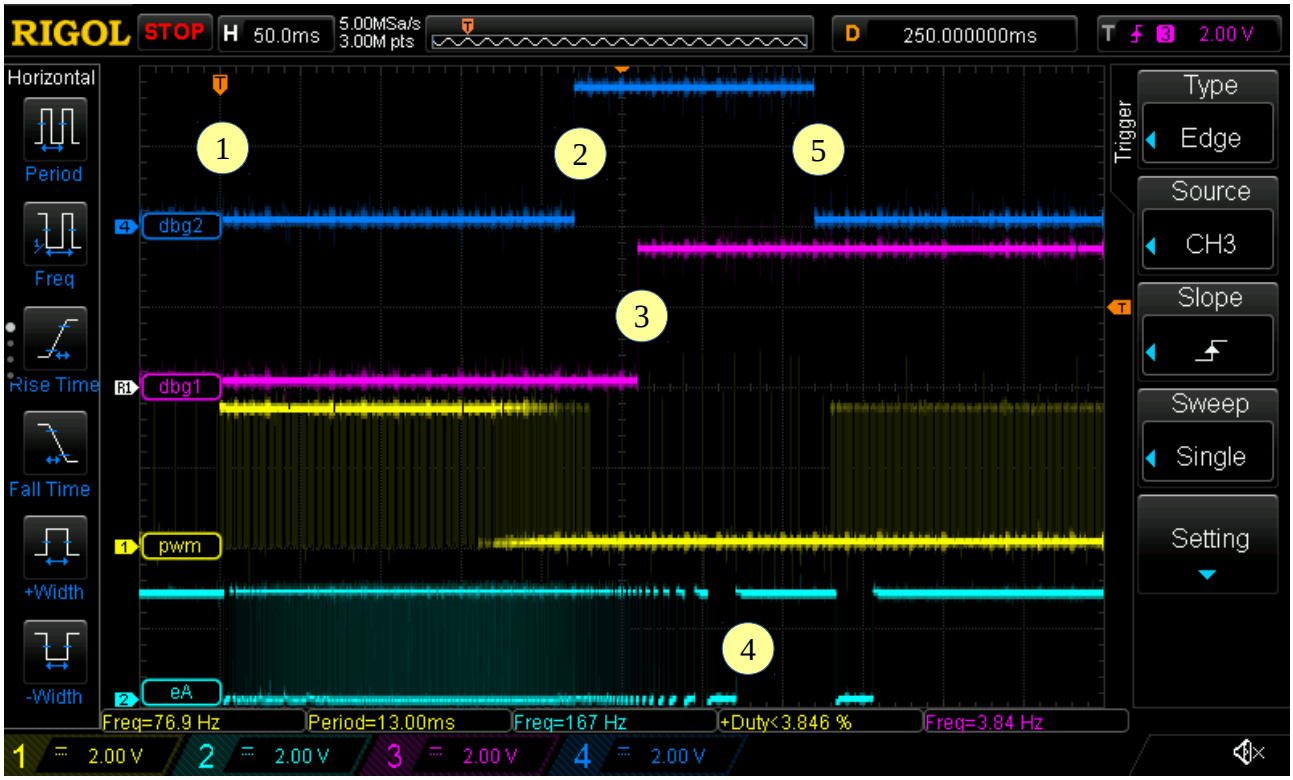
Description:

1. T=0ms
   First PID controller cycle after calibration.
   Motor shaft starts moving 3.2ms after PWM activation.
2. T=160ms
   Entering stop state due to request to change motor direction.
   Duration of stop state is 150ms.
3. T=200ms
   Motor shaft position is within set point ±1.5%.
   This position keeps stable within limits.
4. T=280ms
   Motor shaft completely stopped 120ms after entering stop state.
5. T=310ms
   Leaving stop state with very low PWM duty, not enough to move motor shaft.

8.3 PID controller test#1.2

Conditions:

- Motor shaft position set point: 0x4c8 = 1224 (270°)



Description:

1. T=0ms
   First PID controller cycle after calibration.
   Motor shaft starts moving 4ms after PWM activation.
2. T=220ms
   Entering stop state due to request to change motor direction.
   Duration of stop state is 150ms.
3. T=260ms
   Motor shaft position is within set point ±1.5%.
   This position keeps stable within limits.
4. T=320ms
   Motor shaft completely stopped 100ms after entering stop state.
5. T=370ms
   Leaving stop state with low PWM duty, just enough to move motor shaft.

9 PID controller test#2

9.1 Conditions

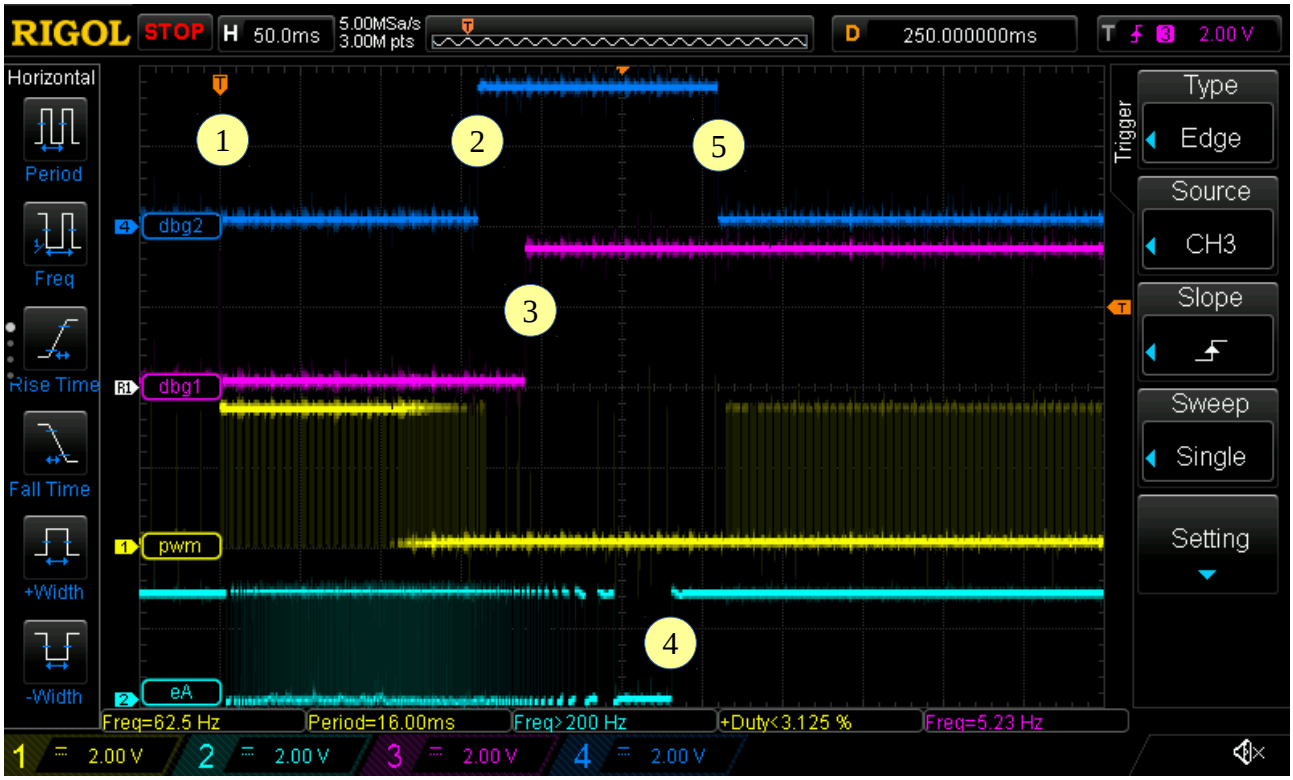
- PID controller frequency: 200Hz (T=5ms)
- PID parameters:

  P=300
  I=0
  D=1650
- Integral part (I) was set to zero. Several experiments showed that adding this component would either slow down the system, or make it unstable. The typical I-values that were tested were in the range of 0.025 – 10.

9.2 PID controller test#2.1

Conditions:

- Motor shaft position set point: 0x330 = 816 (180°)



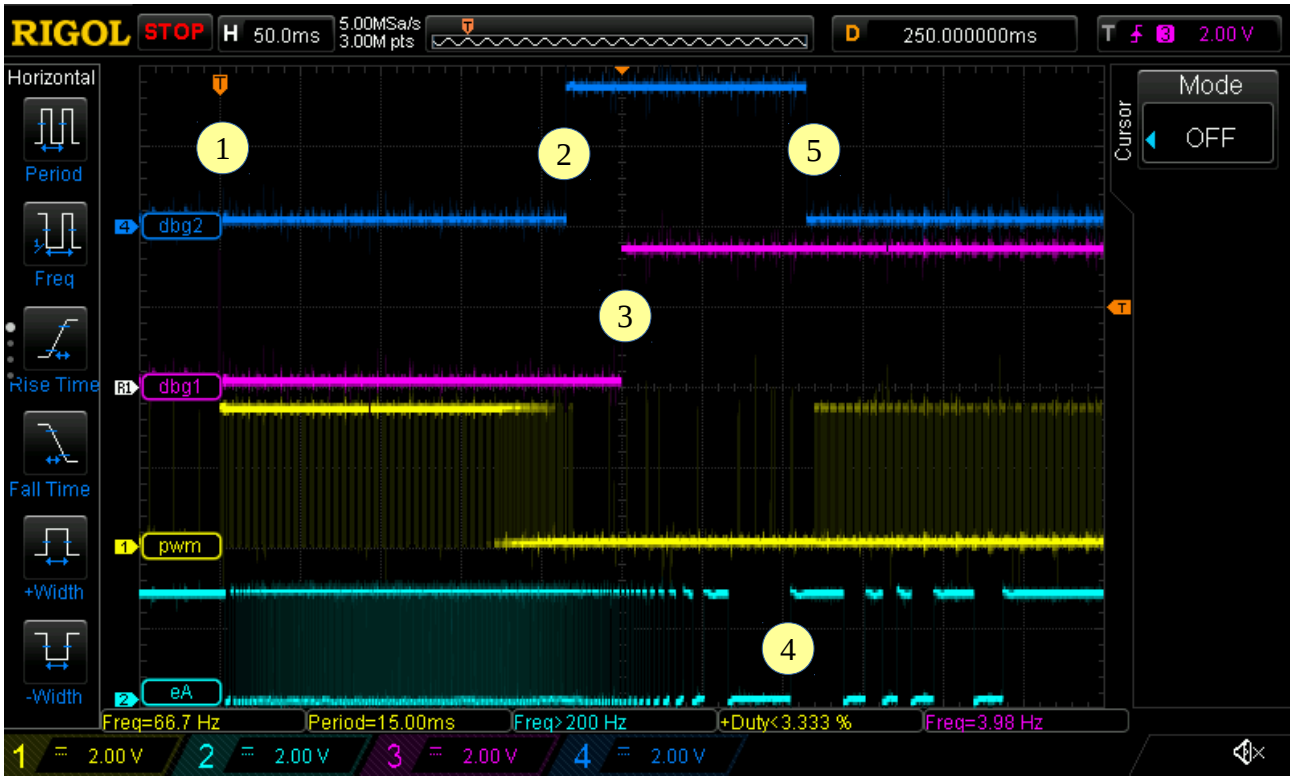
Description:

1. T=0ms
   First PID controller cycle after calibration.
   Motor shaft starts moving 4ms after PWM activation.
2. T=160ms
   Entering stop state due to request to change motor direction.
   Duration of stop state is 150ms.
3. T=190ms
   Motor shaft position is within set point ±1.5%.
   This position keeps stable within limits.
4. T=280ms
   Motor shaft completely stopped 120ms after entering stop state.
5. T=310ms
   Leaving stop state with very low PWM duty, not enough to move motor shaft.

9.3 PID controller test#2.2

Conditions:

- Motor shaft position set point: 0x4c8 = 1224 (270°)



Description:

1. T=0ms
   First PID controller cycle after calibration.
   Motor shaft starts moving 4ms after PWM activation.
2. T=215ms
   Entering stop state due to request to change motor direction.
   Duration of stop state is 150ms.
3. T=250ms
   Motor shaft position is within set point ±1.5%.
   This position keeps stable within limits.
4. T=355ms
   Motor shaft completely stopped 100ms after entering stop state.
5. T=365ms
   Leaving stop state with low PWM duty, just enough to move motor shaft.