# ES-ESP5200 Modern Embedded Systems programming

# Coursework 2017 / 2018

# Designing, Developing and Testing an Embedded Level Controller

Name: **Atle Husmo Undrum**
Partner name: **Torstein Gaarder**

The embedded application we have implemented is a cascade based level regulation of a fluid in a tank.

## Part A

### 1. Introduction

We are developing a level controller based on a cascade regulation of flow into the tank. This tank has a manual control of the flow out of the tank. The cascade regulation is based on two sensors, one for the primary process variable and a second one for the secondary process variable. And a PID regulation is based on discrete time steps calculations. The regulation system will also have an alarm function for high and low level in the tank based on two ON and OFF switches. This is the asynchronous aspect with highest priority in the system. The general time discrete PID regulation is given by this formula:

$$\boldsymbol{u(t_c) = u_p(t_c) + u_i(t_c) + u_d(t_c)} \rightarrow$$

$$\boldsymbol{u_p(t_c) = K_p e(t_c)}$$

$$\boldsymbol{u_i(t_c) = u_i(t_{c-1}) + \frac{K_p T_s}{T_i} e(t_c)}$$

$$\boldsymbol{u_d(t_c) = K_p T_d \frac{e(t_c) - e(t_{c-1})}{T_s}}$$

$$\boldsymbol{e(t_c) = y_{SP}(t_c) - y_{mf}(t_c)}$$

Where $\boldsymbol{u(t_c)}$ is the controller output, $\boldsymbol{u_p(t_c)}$ is the Proportional element to the output controller, $\boldsymbol{u_i(t_c)}$ is the Integral element to the output controller and $\boldsymbol{u_d(t_c)}$ is the Derivative element to the output controller. $\boldsymbol{K_p}$ is the proportional Gain factor. $\boldsymbol{T_s}$ is the size if a time step, $\boldsymbol{t_c}$ is the time step, and $\boldsymbol{t_{c-1}}$ is the previous time step. The variable $\boldsymbol{e}$ is the regulation error(offset) between Process value and Setpoint value. And $\boldsymbol{T_d}$ is the derivate time, and $\boldsymbol{T_i}$ is the integral time. $\boldsymbol{y_{SP}}$ is the process Setpoint and $\boldsymbol{y_{mf}}$ is the measured and filtered process value. There is also a LCD display for indicating system values as setpoint, current value both level and flow and output value to the pump. These are indicated in percent of the measured values. Figure 1 shows a normal PID regulator, while Figure 2 shows the specific PID implementation we are using.

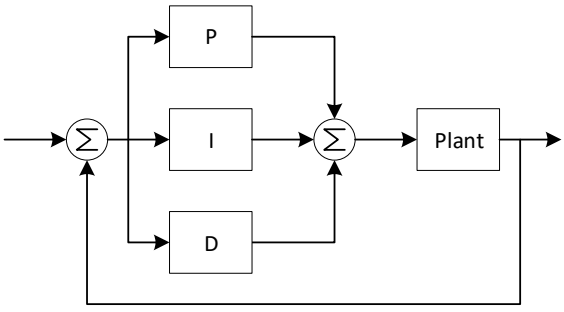


*Figure 1 General PID*

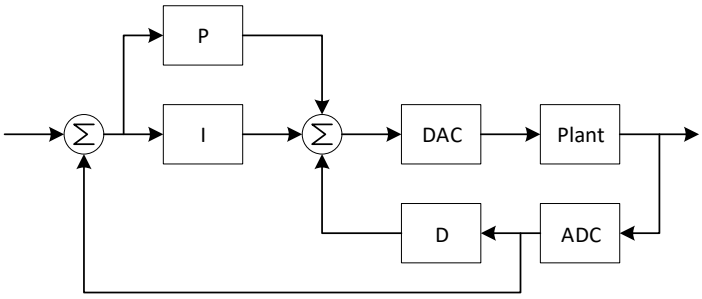


*Figure 2 Out PID implementation*

## 2. Design and Development

The system developed are a regulation system with cascade and PID regulation. The hardware consists of a Pump (P-101), two analogue transmitters (LT-101 and FT-101), two level switches (LAH-101 and LAL-101), a pushbutton for resetting the level alarms (LAR-101), two LEDs for indicating level alarms (LAIH-101 and LAIL-101), a tank (T-101), also a LCD display for showing process values (I-101) and a manual valve for manipulating flow out of the tank (V-201). And to control all this an Atmel ATmega 2560 chip placed on an Arduino Mega Board, this chip also includes the flow controller (FC-101) but is called Level controller (LC-101) this is due to that it is the level in the tank that the system is supposed to control.

The general equation for discrete PID regulation is given in the introduction. But in the cascade regulation the flow control gets its setpoint from the level controller and adjust accordingly. This gives an effect that can be seen in the display there the value of the flow transmitter and the value of the output to the pump should be close in value to each other. Figure 3 shows the P&ID for the system.

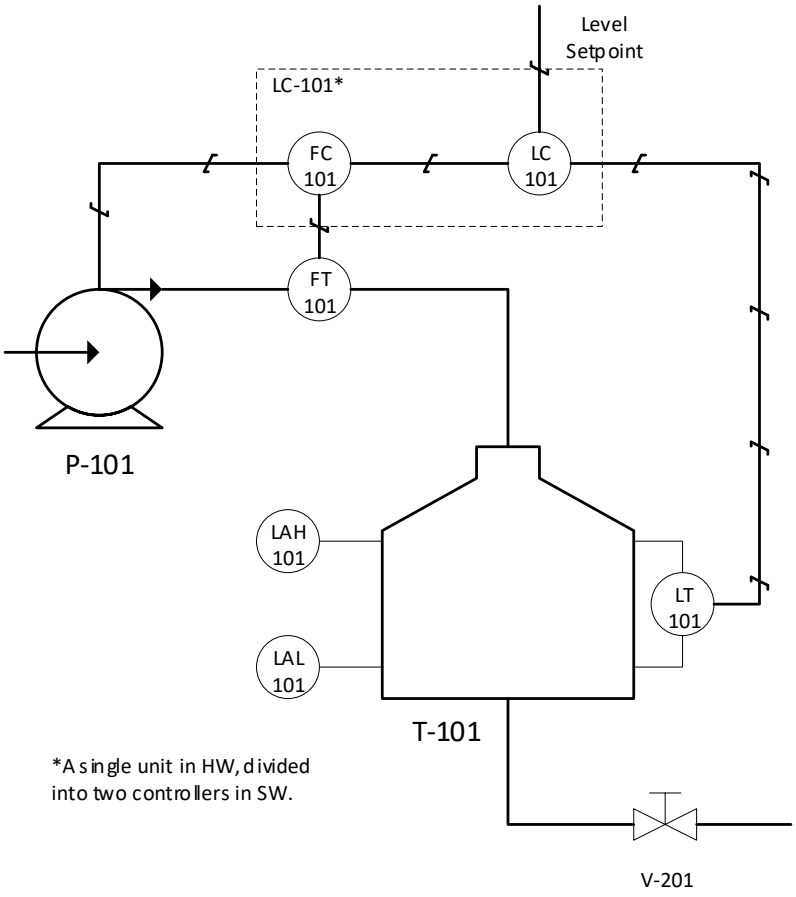


*Figure 3 P&ID*

The software program for the Atmel ATmega 2560 consists of several parts. The code is divided into files (libraries) from 3rd party supplier, license file and the main code. The 3rd party supplier files are:

- I2cmaster.h
- Twimaster.c
- Pcf8574.h
- Pcf8574.c
- Lcdpcf8574.h
- Lcdpcf8574.c

In these files the setup of the TWI (I2C) bus is configured. It is necessary to set the address of the slave (0x27) in the pcf8574.h file and the values of the LCD parallel ports in the lcdpcf8574.h file.

The main.c file is also divided into several parts. These are:

- Includes and definitions
- Functions
    - InitialiseGeneral
    - InitialiseTimer1
    - InitialiseADC
    - InitialiseTimer3_FastPWM_Single
    - Initialise_HW_Interrupts
    - UpdatePID
- Structs
    - PID FlowController
    - PID LevelController
- Variables
- Interrupt Service Routines (ISR)
    - ISR(ADC_vect)
    - ISR(TIMER1_COMPA_vect)
    - ISR(INT4_vect)
    - ISR(INT5_vect)
- Main
    - Main(void)

Figure 4 shows the structure of the program with IO and data flow.

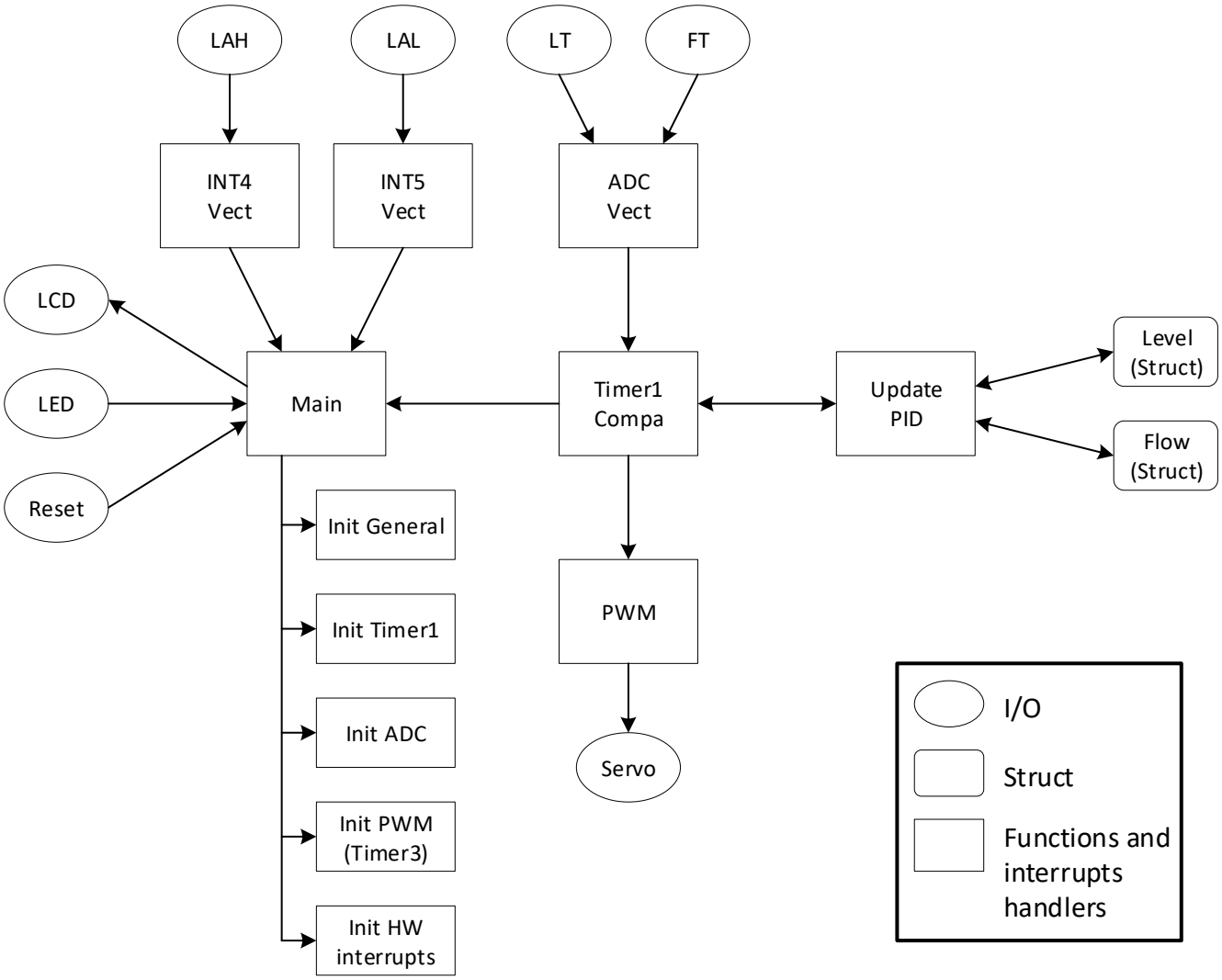


*Figure 4 Program structure*

## Function explanation

As seen over the program has several functions and parts, here a short explanation of each of them follows:

- The main is responsible for writing to the LCD display, trigger the Alarm LED when LAH or LAL has been triggered and to reset the alarms. It's also responsible for initializing the initializers. And to scale the variables for the LCD output.
- The InitialiseGeneral is responsible for initialising //stuff which doesn't go in other initialisations. // Like port declaration and enable interrupts.
- The InitialiseTimer1is responsible for initialising an interrupt on a one second interval //Copied from TimerDemo3.
- The InitialiseADC is responsible for initialising the ADC inputs.
- The InitialiseTimer3_FastPWM_Single is responsible for initialising the PWM for the output to the pump (P-101)
- The Initialise_HW_Interrupts is responsible for initialising the hardware interrupts.
- The ISR(ADC_vect) is the interrupt handler for the ADC channels.
- The ISR(TIMER1_COMPA_vect) is the interrupt handler for the timer 1 for the PID regulator, it's also responsible for scaling and updating the register (OCR3A) for the PWM.
- The ISR(INT4_vect) is the interrupt handler for the LAH.
- The ISR(INT5_vect) is the interrupt handler for the LAL.
- The UpdatePID is the PID controller and it is running inside the ISR(TIMER1_COMPA_vect)
- And the struct PID is for storing PID values

Figure 5 shows the flow of most of the program functions.

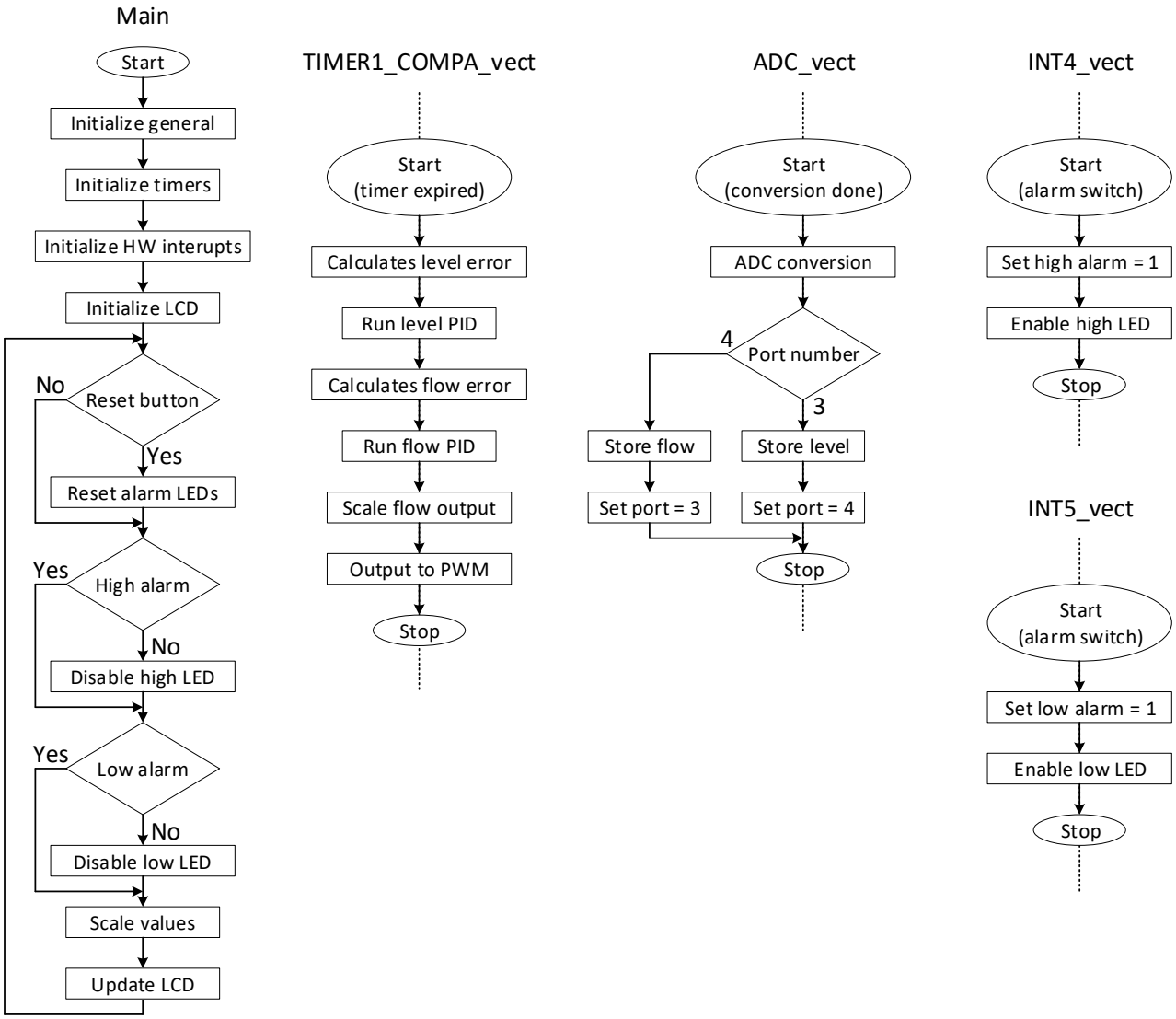


*Figure 5 Program flow*

## I/O List

| I/O List for casade regulation with Arduino Mega with Atmel ATmega 2560 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Inputs | | | | | | | | |
| ID-Tag | Arduino-ID-Tag | Arduino-Port | Atmega Port | Atmega IO Type* | Function | Device tag | Device port | Device Description |
| | | | | | | | | |
| LT-101-S | LC-101 | A-3 | PF3 | ADC | Level Transmitter | LT-101 | FT-101-Signal | Analoge level transmitter |
| FT-101-S | LC-101 | A-2 | PF2 | ADC | Flow Transmitter | FT-101 | LT-101-Signal | Analoge level transmitter |
| LAH-101-S | LC-101 | PWM-2 | PE4 | INT | Level Alarm High | LAH-101 | LAH-101-Signal | Level Switch |
| LAL-101-S | LC-101 | PWM-3 | PE5 | INT | Level Alarm Low | LAL-101 | LAL-101-Signal | Level Switch |
| LAR-101-S | LC-101 | DI-49 | PL0 | DI | Level Alarm Reset | LAR-101 | LAR-101-Signal | Reset Pushbutton |
| | | | | | | | | |
| | | | | | | | | |
| Outputs | | | | | | | | |
| ID-Tag | Arduino-ID-Tag | Arduino-Port | Atmega Port | Atmega IO Type* | Function | Device tag | Device port | Device Description |
| | | | | | | | | |
| P-101-S | LC-101 | PWM-5 | PE3 | PWM | Pump | P-101 | P-101-SG90-PWM | Fluid Pump |
| I-101-D | LC-101 | COMM-SDA-20 | PD1 | TWI-Data | LCD-Data | I-101 | I-101-FC-113-SDA | LCD Display |
| I-101-C | LC-101 | COMM-SCL-21 | PD0 | TWI-Clock | LCD-Clock | I-101 | I-101-FC-113-SCL | LCD Display |
| LAIH-101-S | LC-101 | DI-22 | PA0 | DO | LED | LAIH-101 | LAIH-101-A | LED Light |
| LAIL-101-S | LC-101 | DI-23 | PA1 | DO | LED | LAIL-101 | LAIL-101-A | LED Light |
| | | | | | | | | |
| | | | | | | | | |
| *More About Control Registers and their values in Sheet2 | | | | | | | | |

## Control Registers

Atmega 2560 Control Registers and their values and expected control range

| Ports | PF2 and PF3 | |
|---|---|---|
| Atmega IO Type* | ADC | |
| Register | Enforcing type and Value | Comment |
| ADMUX | = 0b01100010; | // AVCC REF, Left-adjust output (Read most-significant 8 bits via ADCH), Convert channel 2 |
| | '= 0b01100011; | // AVCC REF, Left-adjust output (Read most-significant 8 bits via ADCH), Convert channel 3 |
| ADCSRA | = 0b10101101; | // ADC enabled, Auto trigger, Interrupt enabled, Prescaler = 32 |
| ADCSRB | = 0b11110000; | // clear bits 3,2,1,0 (Free running mode) |
| DIDR0 | = 0b00001100; | // Disable digital input on bits 2 and 3 |
| DIDR2 | = 0b11111111; | // Disable digital input on all bits (64-pin version of ATmega1281 does not even have these inputs) |
| ADCSRA | \|= 0b01000000; | // start ADC conversion |

| Ports | PE4 and PE5 | |
|---|---|---|
| Atmega IO Type* | Interrupts | |
| Register | Enforcing type and Value | Comment |
| DDRE | =0xFF | |
| PORTE | =0x00 | |
| EICRA | = 0b00000000; | // INT 3,2 not used, Interrupt Sense (INT1, INT0) falling-edge triggered |
| EICRB | = 0b00001010; | // INT7 ... 4 not used |
| EIMSK | = 0b00110000; | // Enable INT1, INT0 |
| EIFR | = 0b00110000; | // Clear INT1 and INT0 interrupt flags (in case a spurious interrupt has occurred during chip startup) |

| Ports | PL0 | |
|---|---|---|
| Atmega IO Type* | DI | |
| Register | Enforcing type and Value | Comment |
| DDRL | =0x00 | |
| PORTL | =0x00 | |

| Ports | PE3 | |
|---|---|---|
| Atmega IO Type* | PWM | |
| Register | Enforcing type and Value | Comment |
| TCCR3A | = 0b10000010; | // Fast PWM non inverting, ICR3 used as TOP |
| TCCR3B | = 0b00011001; | // Fast PWM, Use Prescaler '1' |
| TCCR3C | = 0b00000000; | |
| ICR3 | = 25000; | //Sets PWM cycle pulse to 25mS |
| TCNT3H | = 0; | // 16-bit access (write high byte first, read low byte first) |
| TCNT3L | = 0; | |
| OCR3A | = 1750; | //Sets PWM width to 1750uS |
| OCR3A(Highest) | =2500 | //Datasheet says between 1-2ms |
| OCR3A(Lowest) | =1000 | //Needs to be fine tuned |
| TIMSK3 | = 0b00000000; | // No interrupts needed, PWM pulses appears directly on OC3A, OC3B (Port E Bits 3,4) |
| TIFR3 | = 0b00101111; | // Clear all interrupt flags |

| Ports | PD0 and PD1 | |
|---|---|---|
| Atmega IO Type* | TWI(I2C) Master | |
| Register | Enforcing type and Value | Comment |
| TWBR | (Black-box) | TWI Bit Rate register |
| TWCR | (Black-box) | TWI control register |
| TWDR | (Black-box) | TWI Data register |
| TWSR | (Black-box) | TWI Status Register |

| Ports | PA0 and PA1 | |
|---|---|---|
| Atmega IO Type* | DO | |
| Register | Enforcing type and Value | Comment |
| DDRA | 0xFF | |
| PORTA | 0x00 | |

## Timing requirements

The timing requirement for the system aren't that complex, but they are important for proper operations of the system, namely cascade level control in a single tank. Figure 6 shows the sequence diagram for the system, while Figure 7 shows the timing diagram.

### Prioritization

There will be four interrupts with priority in the system. Level alarm high, level alarm low, PID calculations, and analogue to digital conversion. They will be implemented to get priorities to match the order of this list, with level alarm high getting the highest priority. A PWM timer will also be used, but because it won't use the CPU, its priority isn't that important to consider.

### Scheduling

There will be no scheduling beyond using interrupts. In this system, the use of interrupts should be more than enough to guarantee proper function.

**Event periodicity**
The PID calculations must be periodic to get a proper result and the PWM must be periodic to be able to properly control the servo. Making the PWM periodic won't be a challenge as it is a hardware timer which automatically resets and little or no impact from the CPU. Making the PID periodic should also be strait forward, as the only other interrupt which triggers during normal operations has a lower priority, and the period is quite long at one second. If the time to calculate the results is less than one second, the PID should stay periodic, and PID calculations are not very resource intensive to do.

**WCET**
The worst-execution time for the PID controllers must be less than one second. This will be achieved by having everything it relies on controlled by a single timer interrupt. No other timers used will have higher priority. The only parts of the system with higher priority will be the alarm inputs, and when they are triggered, something is wrong with the levels.

The only other parts of the system which cares about execution time is the PWM, which doesn't use CPU, and the ADC, which only uses a small amount of CPU when it has completed a conversion.

Updating the LCD is a slow process which will take a long time, but it's not in any way critical for the operation of the system, so it's not necessary to worry about its WCET.

**Programmable timers.**
Only two programable timers are used, timer 1 and timer 3.

- Timer 1 is set up with an output compare march interrupt. This interrupt causes the PID calculations to run and the output to be updated. The exact duration this timer should be tuned for the individual application, but it is important that it is periodic.
- Timer 3 is set up as fast PWM. As such it doesn't need to generate an interrupt as the PWM automatically outputs to Port E bit 3 & 4.

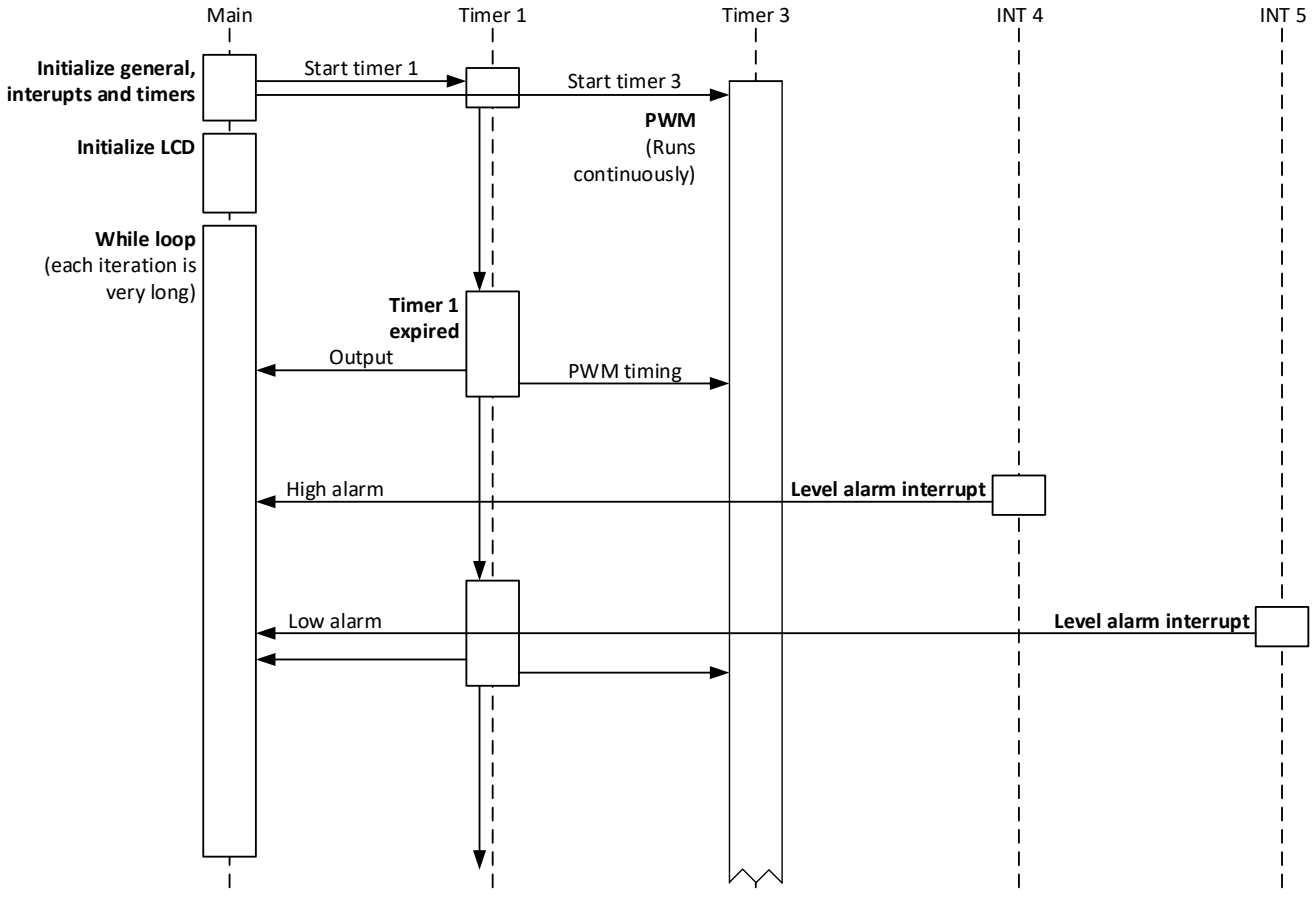


*Figure 6 Sequence diagram*

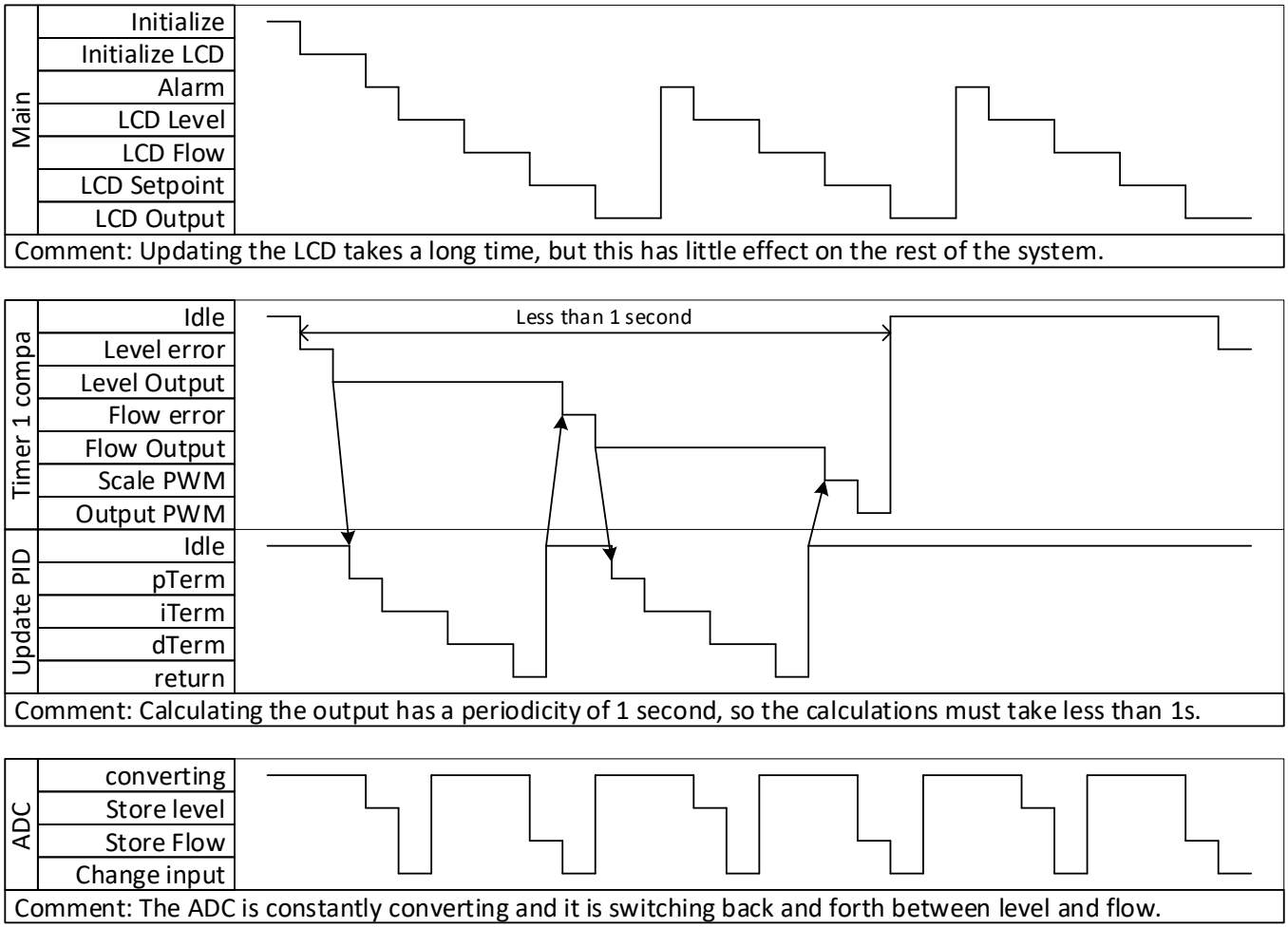


*Figure 7 Timing diagram*

## 3. Testing

To test this system there has been developed a Test Plan with functional testing and non-functional testing. Every hardware component has been identified with a Test ID and a pass/fail grade and other comments for further work and improvements. Also, software component has been identified with a Test ID and a pass/fail grade and other comments for further work and improvements. The same goes for Non-functional tests.

**Test plan for casade regulation with Arduino Mega with Atmel ATmega 2560**

| ID-Test-Tag | ID-Tag | Functional testing | P | F | other comments |
|---|---|---|---|---|---|
| | | | | | |
| FT-101-T | FT-101 | Flowtransmitter range | x | | 8 bit ADC |
| | | High = 100% | x | | |
| | | Low = 0% | x | | |
| | | | | | |
| LT-101-T | LT-101 | Leveltransmitter range | x | | 8 bit ADC |
| | | High = 100% | x | | |
| | | Low = 0% | x | | |
| | | | | | |
| I-101-T | I-101 | LCD-display | x | | very slow |
| P-101-T-01 | P-101 | PWM (Angle from -90°..90°) | x | | drains power wich interfer with the LCD display |
| P-101-T-02 | P-101 | Servo | | x | Inaccurate servo unit |
| LAH-101-T | LAH-101 | Level Alarm High | | x | Doesn't work. Need more testing to find the error |
| LAL-101-T | LAL-101 | Level Alarm Low | | x | Doesn't work. Need more testing to find the error |
| LAIH-101-T | LAIH-101 | Level Alarm High LED | | x | Ref. LAH-101-T |
| LAIL-101-T | LAIL-101 | Level Alarm Low LED | | x | Ref. LAL-101-T |
| LAR-101-T | LAR-101 | Level Alarm Reset | | x | Can't be tested before LAH or LAL works |
| | | | | | |
| LC-101-T-01 | Software | PID Calculations | x | | |
| LC-101-T-02 | Software | Edit of P-gain | x | | |
| LC-101-T-03 | Software | Edit of I-time | x | | |
| LC-101-T-04 | Software | Edit of D-time | x | | |
| | | | | | |
| LC-101-T-05 | Software | Interrupt handlers | | | Level alarm int needs more work |
| LC-101-T-06 | Software | ADC Interrupt | x | | |
| LC-101-T-07 | Software | PWM Interrupt | x | | |
| LC-101-T-08 | Software | Level Alarm High Interrupt | | x | Ref. LAH-101-T |
| LC-101-T-09 | Software | Level Alarm Low Interrupt | | x | Ref. LAL-101-T |

| ID-Test-Tag | ID-Tag | Non-functional testing | P | F | other comments |
|---|---|---|---|---|---|
| | | | | | |
| LC-101-T-11 | LC-101 | Enough power to function properly | | x | PWM-Servo and LCD Display drain to much power together |
| LC-101-T-11 | LC-101 | Timing requirements | x | | Ref. Timing Requirements Chapter |
| LC-101-T-12 | LC-101 | Safe to use | x | | Only low voltage, less then 120 VDC or 50 VAC and no mechanical hazards |

**4. Program code listings**
The program code is included as a separate .pdf file.

**Part B**
**5. Critical evaluation and conclusion**

### Evaluation
The system is not properly (finished due to some errors) developed because of several things. These are: power management, interrupt handlers and code maintainability. These problem areas of the system will be explained further. But the overall function cascade regulation function of the system works properly. There are some hardware faults that also should be corrected mainly the servo of this system. Further explanations about these problems and its solutions will be given in a later paragraph.

The development process was an iterative and cooperative one between Torstein and Me. We started the project with a short description including a general PID Equation and a P&ID diagram of the system to be developed. After that we decided the Inputs- and Outputs- ports to the ATmega 2560 chip

Then we chose a code base to start from. This includes an example project and library for the LCD display we had available and a code base for a PID regulation for C code and code provided from class for the PWM output and ADC input.

The coding process was done in an iterative way with rapid prototyping and testing on the actual board configuration. The main work done in coding was the code for the LCD display and the code for the PID regulation itself driven from the ISR(TIMER1_COMPA_vect) and the Struct for PID variables. There was also some work with the output from the PID regulation to the PWM output, here there were some scaling adjustments work to be done. My partner Torstein did most of the regulation part and I did most of the HMI on the LCD part. So, for the coding part it was divided roughly into 50/50 % between each other.

For the documentation parts Torstein did Timing Requirements including figures to the timing Requirements. While I did I/O list and control registers list and test plan. Both of us worked on the P&ID. For the other parts of the report Torstein did the drawings while I did the writing, this was also a roughly divided into 50/50 % between each other.

The testing of the system was done in an iterative manner as mentioned before. This worked well except the LAH-101 and LAL-101 was implemented to late in the process and was not finished before deadline. The parts of the system that works well is the ADC, the LCD, the PID-regulation and the PWM output.

### Problems and Errors
There were some problems in this project. One of them was the servo where due to inaccuracy in the servo module itself the servo did not respond properly to our control signal. The other was the HW interrupts for the Level Alarms. Here a solution might be to change the port for these inputs. And an effect of this was that the LAIH-101, LAIL-101 and LAR-101 was not properly tested. Under the testing another problem occurred. The problem was that the LCD flickered while the servo was adjusting its position. Probable solution to this is to implement extra power into the system or to chose other parts for the system.

### Conclusions
In conclusion the system can be considered a prototype not a finished project. This is due to the problems described in the above paragraph. For future work the Level Alarms, the servo unit, the power usage and development of code libraries should be prioritised. And to test it against an actual test environment.

**Learning outcomes**
The learning outcome of this project has been repetitioning of things and skills from earlier.