FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

182.694 Microcontroller VU

FAKULTÄT FÜR !NFORMATIK

Martin Perner

SS 2017

Featuring Today:

Introduction App 1

Primer Control Theory

Compilation of C programs

Weekly Training Objective

- Already done
  - 2.2.3 LED Rain ∗
  - 2.2.9 LED curtain ∗
  - 2.4.2 Calling conventions I
  - 2.4.3 Calling conventions II
- This week
  - 3.1.1 C demo program †
  - 3.3.2 Interrupts †
  - 3.4.3 Generating periodic signals †
  - 3.8.1 Switches †
- Next week
  - 3.1.3 Floating point operations ∗
  - 3.3.1 Interrupt & callback demo ∗
  - 3.6.1 UART receiver ∗
  - 3.6.2 UART sender ∗

Review of first exam

- Most of you performed well in the first exam (on average about 14 pts).
- 17 of you (about 28%) performed very well (i.e., scored at least 20 pts).
- If you did not do too well in the first exam: You still have all chances to achieve a good grade, since we consider the best two out of three exams when determining your grade.
- Students who done at least one of the bonus tasks performed better (7 vs. 5 pts on average for the practical part)! Also, 40% of this group achieved at least 20 pts in total.
- Problem areas included
  - I/O-registers (lds vs. out),
  - understanding of division by a power of 2,
  - overly complicated Makefiles,
  - Task 2 in general,
  - and overly complicated program structure.

Application 1

- Fan-Speed-Control (Fantastic)
  - Specification Online (soon)
  - Additional hardware will be in the Lab this week (some is already there).
  - We are in the Lab every
    Friday 13:00 – 14:00
    $\rightarrow$ good time for questions
- Specification
  - given module partition
  - given module interfaces
  - should help you split the big chunk into pieces
- Log time!

Time management

Why log time?

- Learn how expectations differ from reality, and account for it (overhead).
- See when it may be a good time to switch to another task.
- This is important in industry!
  - More hours in offer than actually required: good
  - For too many hours in offer: may not win the bid
  - Too few hours in offer: loss of profit and/or unhappy customer
- You can, of course, make more fine-grained estimations compared to the template!
- Be honest, we do not judge (or grade) the time you needed.

Application 1 – Implementation Proposal

The facts

- Optional.
- Due on Sunday at 23:59!
- If you talk with a tutor, in the following week, about the IP you may get up to 2 bonus points.
- Do not forget to upload it in myTI before the deadline!
- Between 3 and 6 pages (excluding title-page and ToC $\rightarrow$ 5–8 pages in your pdf-viewer)
- We can provide the facilities for an anonymous review among students.
  myTI will remove the first page automatically, so be sure that your name is only on the title-page (and there is a title-page to remove)!

IP Review

What should it be done?

- There will be three free-text forms given, for each of the two assigned reviews
  - Appropriateness (= elegance, correctness, completeness) of the solution in general
  - Presentation (= clarity of exposition, appearance) of the solution
  - Comments to the author

IP Review

How should it be done?

- A review should be
  - done properly, which means it takes time.
  - objective.
  - fair.
  - helpful.
- Provide constructive critique.
- Consider other ideas for what they are, and do not think of your way as the only right way.

Application 1

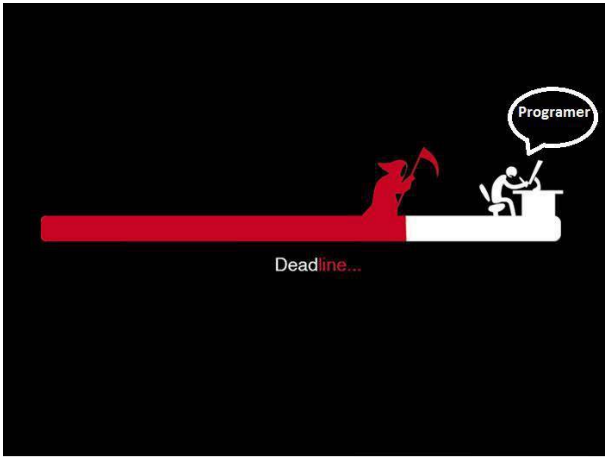


Figure: Deadline versus Programmer. http://devcv.me/2013/08/deadline-versus-programmer

Reading a Datasheet

The basics

- Usually very dense (unlike the ATmega manual)
- Check for new versions
- Many chips are compatible with chips from other vendors; check different manuals or the manuals of the chips used in the module (GLCD!)

Reading a Datasheet

Typical Problems

- Errors in datasheet (new version, alternative datasheet)
- Operation voltage
- Reset conditions; sometimes power-on reset solved in hardware, but later reset have to be done properly
- Unclear, or missing, details
- Important facts that appeared unimportant on first read

Goals

- Short introduction into control theory.
- Only basic knowledge for a rough (and incomplete) understanding of the topic. 376.000 Automation (Automatisierung) provides a better introduction.
- Sufficient knowledge to understand the controller used in the Application.

What is Control Theory?

The Aim

Given a reference value, influence a system in a way such that the output of the system matches the reference.

What is Control Theory?

The Aim

Given a reference value, influence a system in a way such that the output of the system matches the reference.

Open Control Loop (Steuerung)

Given a reference, calculate a control signal which influences the system. This control signal correlates with the output of the system.

What is Control Theory?

The Aim

Given a reference value, influence a system in a way such that the output of the system matches the reference.

Open Control Loop (Steuerung)

Given a reference, calculate a control signal which influences the system. This control signal correlates with the output of the system.

Closed Control Loop (Regelung)

Given a reference, calculate a control signal which influences the system. The output of the system is then measured and incorporated in the calculation of the control signal.

Why do we use Closed Control Loops?

Disadvantages

- The control loop can become unstable.
- A measurement for the feedback is required.

Why do we use Closed Control Loops?

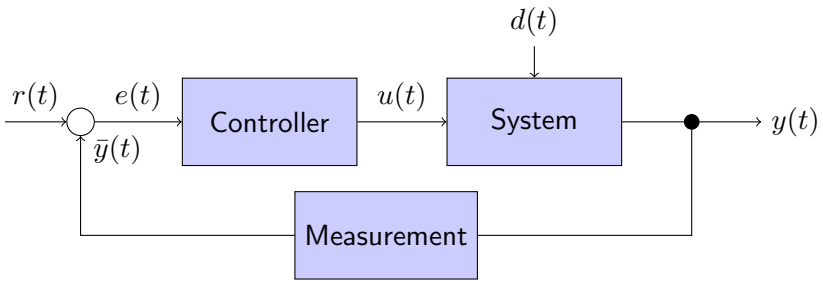
Disadvantages

- The control loop can become unstable.
- A measurement for the feedback is required.

Advantages

- Can reach the reference value, and follow changes automatically.
- Can counteract disturbances acting on the system.

Simple Closed Control Loop Block Diagram



- $r(t)$ reference value
- $e(t)$ error $r(t) - \bar{y}(t)$
- $u(t)$ control signal
- $d(t)$ disturbance
- $y(t)$ system output
- $\bar{y}(t)$ measured system output

Linear Time-Invariant Systems

We restrict ourself to LTI Systems

- Their behaviour can be described with differential equations.
- Solution in time-domain can be challenging.
- Apply Laplace transformation for an algebraic equation in the Laplace domain (frequency).
- Solve and transform back to time domain.

Laplace Transformation

From time to Laplace

$$s = \sigma + j\omega$$

$$\hat{f}(s) = \mathcal{L}\{f(t)\} = \int_0^\infty e^{-st} f(t)\,\mathrm{d}t$$

Laplace Transformation

From time to Laplace

$$s = \sigma + j\omega$$

$$\hat{f}(s) = \mathcal{L}\{f(t)\} = \int_0^\infty e^{-st} f(t)\,\mathrm{d}t$$

From Laplace to time

$$f(t) = \mathcal{L}^{-1}\{\hat{f}(s)\} = \frac{1}{2\pi j}\int_{\sigma-j\infty}^{\sigma+j\infty} e^{st}\hat{f}(s)\,\mathrm{d}s$$

Laplace Transformation

From time to Laplace

$$s = \sigma + j\omega$$

$$\hat{f}(s) = \mathcal{L}\{f(t)\} = \int_0^\infty e^{-st} f(t)\,\mathrm{d}t$$

From Laplace to time

$$f(t) = \mathcal{L}^{-1}\{\hat{f}(s)\} = \frac{1}{2\pi j}\int_{\sigma-j\infty}^{\sigma+j\infty} e^{st}\hat{f}(s)\,\mathrm{d}s$$

Usually correspondence tables are used.

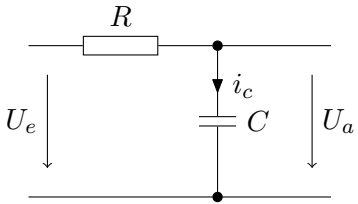
Laplace Transformation

A short correspondence table

time-domain	Laplace-domain
$\sigma(t)$	$\frac{1}{s}$
e^{-at}	$\frac{1}{s+a}$
$\dot{f}(t)$	$s\hat{f}(s) - f(0)$

$\sigma(t)$ is the unit step function: $\sigma(t \geq 0) = 1, \sigma(t < 1) = 0.$

Example: Low-Pass Filter



$$U_a = \frac{1}{C} \int i_c \, \mathrm{d}t$$

$$U_e = i_c R + U_a$$

$$U_e = \frac{\mathrm{d}U_a}{\mathrm{d}t} RC + U_a$$

$$U_e(t) = \dot{U}_a(t) RC + U_a(t)$$

Solution can be non-trivial for certain signals/states/systems!

Example: Low-Pass Filter

$$\begin{aligned}\mathcal{L}\{U_e(t) &= \dot{U}_a(t)RC + U_a(t)\}\\ U_e(s) &= sU_a(s)RC + U_a(s)\\ U_a(s) &= \frac{U_e(s)}{sRC+1}\end{aligned}$$

The transformation back into the time-domain can also be tricky and sometimes requires a partial fraction decomposition.

Example: Low-Pass Filter

$$\begin{aligned}\mathcal{L}\{U_e(t) &= \dot{U}_a(t)RC + U_a(t)\}\\ U_e(s) &= sU_a(s)RC + U_a(s)\\ U_a(s) &= \frac{U_e(s)}{sRC+1}\end{aligned}$$

The transformation back into the time-domain can also be tricky and sometimes requires a partial fraction decomposition.
We use $\sigma(t)$ as $U_e(t)$, restrict $t \geq 0$, and thus

$$U_a(s) = \frac{1}{s(sRC+1)} = \frac{1}{s} - \frac{RC}{RCs+1} = \frac{1}{s} - \frac{1}{s+\frac{1}{RC}}$$

$$U_a(t) = 1 - e^{-\frac{t}{RC}}$$

Transfer Function

Transfer Function $G(s)$

The transfer function is defined as the ratio of the output signal $\hat{y}(s)$ to the input signal $\hat{u}(s)$, $G(s) = \frac{\hat{y}(s)}{\hat{u}(s)}$.

In the time domain, the transfer function maps $u(t) \rightarrow y(t)$.

Transfer Function

Transfer Functions of LTI-Systems

- Linearity
  $a \cdot u_1(t) + b \cdot u_2(t) \rightarrow a \cdot y_1(t) + b \cdot y_2(t)$
- Time invariance
  $u(t - t_0) \rightarrow y(t - t_0)$

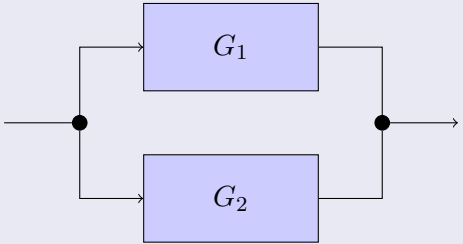
Systems rarely are perfectly LTI, but the approximation is useful.

Transfer Function Building Blocks

- Proportional element (P)
  $x(t) = K_p u(t)$ $\quad G(s) = K_p$
- Integrating element (I)
  $x(t) = K_i \int u(t)\,\mathrm{d}t$ $\quad G(s) = \frac{K_i}{s}$
- Differential element (D)
  $x(t) = K_d \frac{\mathrm{d}u(t)}{\mathrm{d}t}$ $\quad G(s) = K_d s$
- first order delay (PT_1)
  $T_0 \dot{x}(t) + x(t) = K_p u(t)$ $\quad G(s) = \frac{K_p}{1+sT_0}$
- second order delay (PT_2)
  $T_2 \ddot{x}(t) + T_1 \dot{x}(t) + x(t) = K_p u(t)$ $\quad G(s) = \frac{K_p}{1+sT_1+s^2T_2}$

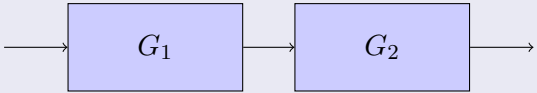
Composition of Transfer Function

Parallel



$$G(s) = G_1(s) + G_2(s)$$

Serial



$$G(s) = G_1(s) \cdot G_2(s)$$

Composition of Transfer Function

Feedback

$$G(s) = \frac{G_1(s)}{1+G_1(s)}$$

Determination of Transfer Function

Mathematical

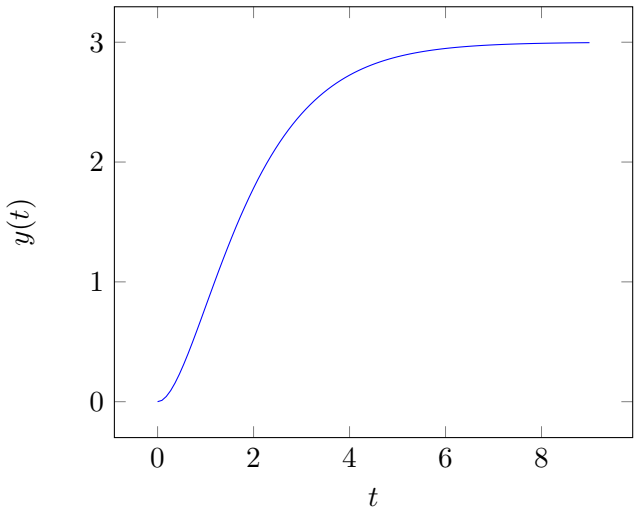
Describe the system with ODE and solve them. Requires knowledge of the system, and may be more detailed than required.

Experimental

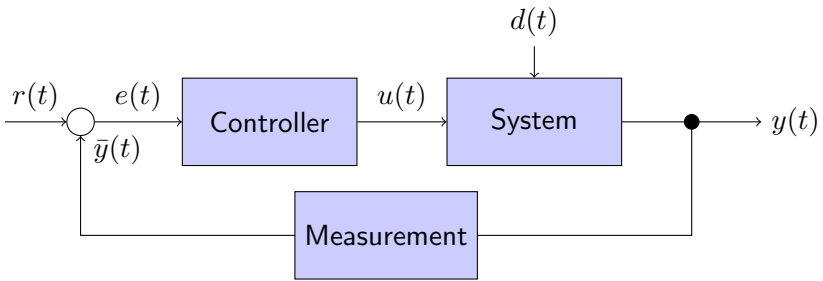
Apply a test function to the input of the open system, record the output, and reconstruct transfer function.
Typical test functions are the dirac delta function ($\delta(0) = \infty$, $\delta(t \neq 0) = 0$), or the unit step function ($\sigma(t \geq 0) = 1$, $\sigma(t < 0) = 0$).

Determination of Transfer Function - Step Response

PT_2 with $T_1 = T_2 = 1$, $K_p = 3$, with $\sigma(t)$ as input



Simple Closed Control Loop Block Diagram



- $r(t)$ reference value
- $e(t)$ error $r(t) - \bar{y}(t)$
- $u(t)$ control signal
- $d(t)$ disturbance
- $y(t)$ system output
- $\bar{y}(t)$ measured system output

Controller Types

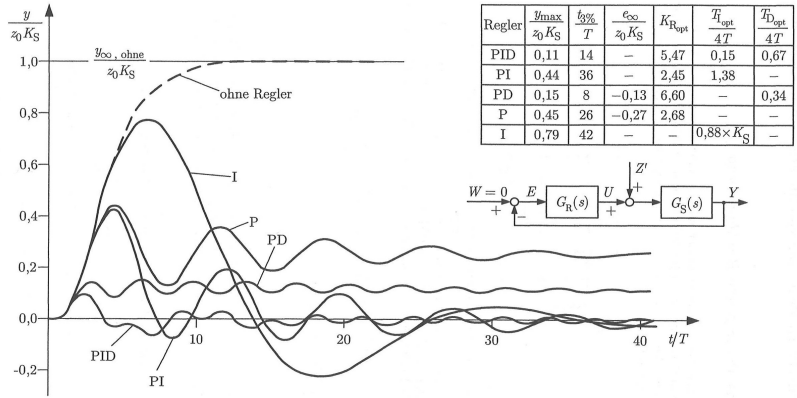
Objective of a Controller

- Minimize the error between the reference and the (measured) system output,
- and do it in a timely manner (fast).
- Ideally, no overshooting (depending on the system, overshooting may not be allowed)

Components of a Controller

In classic control theory, compositions of P, I, D, and first-order delay (T_1) are usually used.

Reaction of Controllers to disturbance



Regler	$\frac{y_{max}}{z_0 K_S}$	$\frac{t_{3\%}}{T}$	$\frac{e_\infty}{z_0 K_S}$	$K_{R_{opt}}$	$\frac{T_{I_{opt}}}{4T}$	$\frac{T_{D_{opt}}}{4T}$
PID	0,11	14	–	5,47	0,15	0,67
PI	0,44	36	–	2,45	1,38	–
PD	0,15	8	−0,13	6,60	–	0,34
P	0,45	26	−0,27	2,68	–	–
I	0,79	42	–	–	$0{,}88 \times K_S$	–

Bild 5.3.4. Verhalten der normierten Regelgröße $y/(z_0 K_S)$ bei sprungförmiger Störung $z' = z_0\sigma(t)$ am Eingang der Regelstrecke $[G_S(s) = K_S/(1+Ts)^4;\ K_S = 1\,]$, die mit den verschiedenen Reglertypen zusammengeschaltet wurde.

Figure: From Regelungstechnik I, H. Unbehauen

Controller Parameterization

With a given Transfer Function

Analytically determine the parameters to get a behaviour within the required settings.

Controller Parameterization

Experimental

Without a defined system, one can resort to

- Trail'n'Error tuning.
- Ziegler-Nichols method
  Using a P-controller, increase the gain (K_p) until the system oscillates. Use a table to determine controller parameters based on K_p and the period of the oscillation. System has to be able to oscillate safely!
- Determine the unit-step response of the system and measure
  - The reached amplitude K
  - The point in time when the tangent of the responses intersects with 0 (dead-time time).
  - The time between the intersection of the tangent with 0 and the intersection with K (time constant).

  There are different tables which allow to achieve certain goals (e.g., limited overshoot) by Ziegler and Nichols, and by Chien, Hrones, and Reswick.

Stability

What is Instability?

A change to the system (disturbance) or reference causes a change of the control signal which does not converge (oscillation).

Potential to destroy the system!

Causes for Instability

- Dead-time
- used Controller type unfitting for the system
- Incorrect parameterization of the controller

Stability

How to detect if a System is unstable?

- There are multiple criteria.
- Usually work with the real and imaginary part of the transfer function of the open system.
- As the resulting plots are not covered today, only note that there are ways to determine if the system is stable.

Digital Controller

What changes

continuous time $\Rightarrow$ discrete time with samples every T

- Laplace transformation does not work anymore!
- z-Transformation can be used (related to Laplace)
- for sufficiently fast controller operation, the difference is negligible
- T needs to be as equidistant as possible, otherwise the system can become unstable!

Digital PID-Controller

Definition

$u(t) = K_p e(t) + \frac{K_p}{T_I} \int_0^t e(\tau)\,\mathrm{d}\tau + K_p T_D \frac{\mathrm{d}e(t)}{\mathrm{d}t}$ or

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

Pseudo-Code

With T being the sample-rate and K_p, K_i, and K_d being the respective controller parameters:

```
integral = 0, prev_e = 0;
while (TRUE) {
  sample e;
  integral = integral + e * T;
  derivate = (e - prev_e) / T;
  u = K_p * e + K_i * integral + K_d * derivate;
  prev_e = e;
  wait for T;
}
```

Digital PID-Controller – Issues

Windup

Due to the definition of the integral, we will see an overshoot. This effect can be limited by various means, e.g., artificially limiting `integral`.

Limited control signal

The output has a limited value range, thus capping u may be necessary.

Arithmetic overflow

Especially large integral parts can cause overflows in the calculations. Usually, the anti-windup is also used to prevent this. Otherwise saturated multiplication/addition could be used.

Discontinuous Controller

Bang-Bang Controller (Zweipunktregler)

The controller follows a hysteresis and only provides a 0/1 output.

- Easy to implement.
- System must be able to handle this hard on/off switches.
- The error will constantly oscillate.
- Size of error depends on the size of the hysteresis.

Questions?

Exercise 3.4.3 “Generating periodic signals”

- Please pay special attention to Exercise 3.4.3.
- Probably the single most important exercise in Part 2
  - Especially if you are new to microcontroller programming.
  - May keep you busy this week.
- **Carefully** read the relevant sections in the ATmega1280 manual.
- Be precise: “25 µs” means “**exactly** 25 µs”.
  - Be as precise as the microcontroller allows you to.
- Use the oscilloscope!
- Exercise 3.4.3 introduces you to the business of interacting with the “real world”.

Overview of today's lecture

- Assembler instruction are (usually) in a one-to-one correspondence with machine opcodes.
- In the first part of the course, we studied ATmega1280's instruction set (i.e., its set of opcodes).
  - No program can use **anything** else than these instructions.
- When programming in a high-level programming language, we ignore the actual sequence of machine instructions generated by the compiler.

Overview of today's lecture

This is good ...

because the code is independent of the target machine.

Overview of today's lecture

This is good . . .

because the code is independent of the target machine.

This is bad . . .

because we lose information.

Overview of today's lecture

The C Programming Language

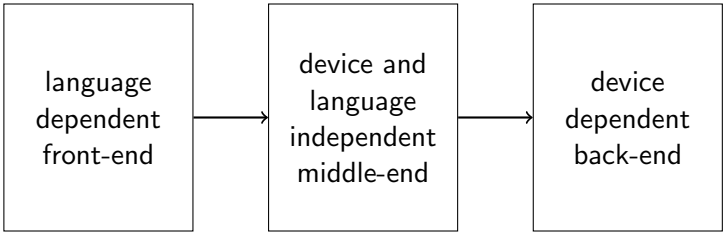
- C is a popular and convenient programming language.
- Coming from the "assembler world", we can infer much knowledge about how C programs end up in machine code.
- Due to the limited hardware resources in microcontroller chips, this knowledge is important to us.
- In particular: We have to know when to tune things manually by inserting assembler code into C programs.

Overview of today's lecture

- Today's lecture introduces you to the business of analyzing compiled C programs and spotting weak points.
- Of course, this analysis will also provide useful insights for everyday C programming.

Our compiler: GCC

- In the lab we use the GNU Compiler Collection.
- GCC has a modular structure:



Our compiler: GCC

- GCC is highly configurable:
  - More than 100 command-line options are concerned with code optimization alone.
  - Check our demo Makefile for various examples!
- GCC is verbose:
  - You can observe every single intermediate step of compilation.
- This makes GCC very suitable for the analysis of the compilation process.

An example – C

```c
#include <avr/io.h>
#define N 5

int main(void) {
    uint8_t i = 1;
    uint8_t x = 0;

    for(; i <= N; ++i) {
        x += i;
    }

    for(;;);
    return 0;
}
```

An example – Assembler

Generated by executing:

```
avr-gcc -mmcu=atmega1280 -S -o test.s test.c
```

```
    .file "test.c"
__SREG__ = 0x3f
__SP_H__ = 0x3e
__SP_L__ = 0x3d
__tmp_reg__ = 0
__zero_reg__ = 1
    .text
.global main
    .type main, @function
main:
    push r29
    push r28
    rcall .
    in r28,__SP_L__
    in r29,__SP_H__
/* prologue: function */
/* frame size = 2 */
/* stack size = 4 */
.L__stack_usage = 4
    ldi r24,lo8(1)
    std Y+1,r24
    std Y+2,__zero_reg__
    rjmp .L2
.L3:
    ldd r25,Y+2
    ldd r24,Y+1
    add r24,r25
    std Y+2,r24
    ldd r24,Y+1
    subi r24,lo8(-(1))
    std Y+1,r24
.L2:
    ldd r24,Y+1
    cpi r24,lo8(6)
    brlo .L3
.L4:
    rjmp .L4
    .size main, .-main
```

GCC passes

1. Parse
2. Create abstract syntax tree (AST)
3. Create GIMPLE
4. Optimize GIMPLE
5. Create register transfer level (RTL) list
6. Optimize RTL
7. Create list of assembler instructions
8. Assemble
9. Link

An example – GIMPLE

Generated by executing:

```
avr-gcc -mmcu=atmega1280 -fdump-tree-gimple test.c
```

```
main ()
{
  int D.1989;
  uint8_t i;
  uint8_t x;

  i = 1;
  x = 0;
  goto <D.1249>;
  <D.1248>:
  x = x + i;
  i = i + 1;
  <D.1249>:
  if (i <= 5)
    goto <D.1248>;
  else
    goto <D.1250>;
  <D.1250>:
  <D.1251>:
  goto <D.1251>;
  D.1989 = 0;
  return D.1989;
}
```

Try: `avr-gcc -mmcu=atmega1280 -fdump-tree-all test.c`

Machine-dependent backend

- Mapping from ASTs (Abstract Syntax Tree) to sequences of assembler instructions.
- Defined in "Machine description" files.
  - e.g., `avr.md`
  - Written in Lisp.
- Example:

```
; return
(define_insn "return"
  [(return)]
  "reload_completed && avr_simple_epilogue ()"
  "ret"
   [(set_attr "cc" "none")
    (set_attr "length" "1")])
```

An example – RTL

Generated by executing:
`avr-gcc -mmcu=atmega1280 -fdump-rtl-expand test.c`
Only showing an excerpt

```
(note 4 2 5 3 [bb 3] NOTE_INSN_BASIC_BLOCK)

(insn 5 4 6 3 test.c:5 (set (reg:QI 42)
        (const_int 1 [0x1])) -1 (nil))


(insn 6 5 7 3 test.c:5 (set (mem/c/i:QI
    (reg/f:HI 37 virtual-stack-vars) [0 i+0 S1 A8])
        (reg:QI 42)) -1 (nil))

(insn 7 6 8 3 test.c:6 (set (mem/c/i:QI
    (plus:HI (reg/f:HI 37 virtual-stack-vars)
                (const_int 1 [0x1])) [0 x+0 S1 A8])
        (const_int 0 [0x0])) -1 (nil))

(jump_insn 8 7 9 3 test.c:8 (set (pc)
        (label_ref 18)) -1 (nil)
 -> 18)
```

Metadata

Set pseudo-register (r44) to value 1.
Store the value of r44 at SP+2

Store 0 at SP+1

Goto label

Optimization

- Optimization trades run time for compilation time (and possibly object file size).
- Five optimization levels (sets of optimization flags) usually available in GCC:
  - -O0 No optimization.
  - -O1 Many optimizations done, excluding instruction scheduling.
  - -O2 More optimizations, some of which increase object file size.
  - -Os Optimizations from O2 except for object file size increasing ones.
  - -O3 More expensive (and object file size increasing) optimizations, in particular function inlining.

Optimization examples

Function inlining

- Paste function body of simple functions instead of calling it.
- Only performed (automatically) at optimization level 3.
- Adding the keyword `inline` to the function declaration suggests the compiler to inline the specific function.
- By using `__attribute__((always_inline))` GCC can be forced to inline a function.
- Is a trade-off between execution speed and code size.
- Code size can be a problem when instruction are cached.
- The compiler may not inline every call to a specific function.

Optimization examples

- Constant folding

```
 in  x = 3 + i - 5;
out  x = i - 2;
```

Optimization examples

- Constant folding

```
 in  x = 3 + i - 5;
out  x = i - 2;
```

- Algebraic simplification

```
 in  x = 2 * y;
out  x = y + y;
```

Optimization examples

- Constant folding

```
 in x = 3 + i - 5;
out x = i - 2;
```

- Algebraic simplification

```
 in x = 2 * y;
out x = y + y;
```

- Common subexpression elimination

```
 in x = x * (t + 5);
    y = y * (t + 5);
out z = t + 5;
    x = x * z;
    y = y * z;
```

Optimization examples

- Dataflow analysis

```
in  x = y + 5;
    z = 42;
    t = x;
out t = y + 5;
```

Optimization examples

- Dataflow analysis

```
in  x = y + 5;
    z = 42;
    t = x;
out t = y + 5;
```

- Jump optimization

```
in                          rjmp blab
                            ...
                          blab:
                            rjmp blub
                            ...
                          blub:
out                         rjmp blub
                            ...
```

Optimization examples

Invariant code motion

- Move loop-independent code in front of loop.
- Makes loops faster!

```
int i, o;
for (i = 0; i < 10; i++) {
  o = y + (z ^ 255);
  m[i] = 2 * i + o;
}
```

```
int i, o = y + (z ^ 255);
for (i = 0; i < 10; i++) {
  m[i] = 2 * i + o;
}
```

Optimization examples

Loop unrolling

- Generally: Copy loop body N times, where N is the number of iterations.
- Saves checks and jumps at iteration boundary.
- Only reasonable if bound on loop iterations is known; or even the exact number
- Loop unrolling and instruction re-scheduling can by used to take advantage of contemporary CPU-pipelines.
- Increases code size.

```
for (i = 0; i < 5; i++) {
  m[i] = 2 * i + o;
}
```

```
m[0] = o;
m[1] = 2 + o;
m[2] = 4 + o;
m[3] = 6 + o;
m[4] = 8 + o;
```

Pointer Arithmetic

While we are on the topic of arrays ...

- How can we define an array?

```
char foo[5] = "abcd";
char bar[] = "abcd";
char *foobar = "acd";
```

- Are there differences?

```
printf("%i", sizeof(foo)); // 5
printf("%i", sizeof(bar)); // 5
printf("%i", sizeof(foobar)); // ?
```

Optimization examples

Function inlining

- Paste function body of simple functions instead of calling it.
- Only performed (automatically) at optimization level 3.
- Adding the keyword `inline` to the function declaration suggests the compiler to inline the specific function.
- By using `__attribute__((always_inline))` GCC can be forced to inline a function.
- Is a trade-off between execution speed and code size.
- Code size can be a problem when instruction are cached.
- The compiler may not inline every call to a specific function.

Inline assembler

- What to do if you discover that the C compiler produces an inefficient piece of code?
  - Just do it yourself!
  - Write a more efficient version (in assembler).
- **Inline assembler** is the buzz-word when replacing compiled code by your own.
- N.B.: Time efficiency is not the most important thing in the world.
  - Time predictability is a major concern.
  - If you write your own assembler code, you control the timing behavior of the program.
  - Often necessary in interrupt service routines.

Inline assembler

- Syntax for AVR inline assembler:

```
asm(code
    : output operand list
    : input operand list
    [: clobber list]);
```

  - "Code" denotes a string that contains assembler code (may also contain labels etc.).
  - "Output operand list" contains output operands.
  - "Input operand list" contains input operands.
  - "Clobber list" is composed of registers that you modified in your inline code, but that are no output registers.
- Let the C compiler write your function prologue and epilogue.

Inline assembler example

```
void delay(uint8_t ms,
      uint16_t delay_count)
{
  uint16_t cnt;
  asm volatile ("\n"
      "L_dl1%=:" "\n\t"
      "mov %A0, %A2" "\n\t"
      "mov %B0, %B2" "\n"
      "L_dl2%=:" "\n\t"
      "sbiw %A0, 1" "\n\t"
      "brne L_dl2%=" "\n\t"
      "dec %1" "\n\t"
      "brne L_dl1%=" "\n\t"
      : "=&w" (cnt)
      : "r" (ms), "r" (delay_count)
      );
}
```

```
;delay_count r23:r22
;ms r24
;delay(ms,delay_count)
delay:
  outer:
    mov r28, r22
    mov r29, r23
    inner:
      sbiw r28,1
      brne inner
    dec r24
    brne outer
```

See Section 11.14 of avr-libc user manual – "What registers are used by the C compiler?"

Inline assembler example

```
void delay(uint8_t ms,
      uint16_t delay_count)
{
  uint16_t cnt;
  asm volatile ("\n"
      "L_dl1%=:" "\n\t"
      "mov %A0, %A2" "\n\t"
      "mov %B0, %B2" "\n"
      "L_dl2%=:" "\n\t"
      "sbiw %A0, 1" "\n\t"
      "brne L_dl2%=" "\n\t"
      "dec %1" "\n\t"
      "brne L_dl1%=" "\n\t"
      : "=&w" (cnt)
      : "r" (ms), "r" (delay_count)
      );
}
```

are they the same?

```
;delay_count r23:r22
;ms r24
;delay(ms,delay_count)
delay:
  outer:
    mov r28, r22
    mov r29, r23
    inner:
      sbiw r28,1
      brne inner
    dec r24
    brne outer
```

See Section 11.14 of avr-libc user manual – "What registers are used by the C compiler?"

Rules of Optimization

The First Rule of Program Optimization: Don't do it.

— Michael A. Jackson

Rules of Optimization

The First Rule of Program Optimization: Don't do it.
The Second Rule of Program Optimization (for experts only!): Don't do it yet.
— Michael A. Jackson

My dear Compiler: What did you do there?

```
char test = 0xff;

if(test >= 130) {
    printf("big\n");
}
else {
    printf("LITTLE\n");
}
```

What is the result?

My dear Compiler: What did you do there?

```
char test = 0xff;

if(test >= 130) {
    printf("big\n");
}
else {
    printf("LITTLE\n");
}
```

What is the result?

It depends.

My dear Compiler: What did you do there?

There are C standards but ...

My dear Compiler: What did you do there?

There are C standards but ... not everything is clear or specified.

My dear Compiler: What did you do there?

There are C standards but ... not everything is clear or specified.

An object declared as type char is large enough to store any member of the basic execution character set. If a member of the required source character set enumerated in 5.2.1 is stored in a char object, its value is guaranteed to be positive. If any other character is stored in a char object, the resulting value is implementation-defined but shall be within the range of values that can be represented in that type. — C99 6.2.5

My dear Compiler: What did you do there?

Signedness of char

Implementation defined.

The output for gcc on x86 and avr is LITTLE (signed char).

My dear Compiler: What did you do there?

Signedness of char

Implementation defined.

The output for gcc on x86 and avr is LITTLE (signed char).

This can be changed with `-funsigned-char` respectively `-fsigned-char`.

My dear Compiler: What did you do there?

```
int o = y + (z ^ 255);
```

What does (z ^ 255) do?

My dear Compiler: What did you do there?

```
int o = y + (z ^ 255);
```

What does (z ^ 255) do?

It depends.

My dear Compiler: What did you do there?

```
int o = y + (z ^ 255);
```

What does (z ^ 255) do?

It depends.

What data-type does z have?
Lets assume it is `int`, then it has ... bits!

My dear Compiler: What did you do there?

```
int o = y + (z ^ 255);
```

What does (z ^ 255) do?

It depends.

What data-type does z have?
Lets assume it is `int`, then it has 16 bits!

My dear Compiler: What did you do there?

```
int o = y + (z ^ 255);
```

What does (z ^ 255) do?

It depends.

What data-type does z have?
Lets assume it is `int`, then it has 16 bits!
So the snipplet negates the lower 8 bits of z.

My dear Compiler: What did you do there?

```
int o = y + (z ^ 255);
```

What does (z ^ 255) do?

It depends.

What data-type does z have?
Lets assume it is `int`, then it has 16 bits!
So the snipplet negates the lower 8 bits of z.

The size of `int` is varies across toolchains/architectures!

Therefore be explicit and use `uint8_t` or `uint16_t` etc.!

My dear Compiler: What did you do there?

Can a compiler be error-free?

gcc: 7+ million LOC
llvm: 2.5+ million LOC

My dear Compiler: What did you do there?

Can a compiler be error-free?

gcc: 7+ million LOC
llvm: 2.5+ million LOC

If debugging is the process of removing bugs, then programming must be the process of putting them in.
— Edsger Dijkstra

My dear Compiler: What did you do there?

Can a compiler be error-free?

gcc: 7+ million LOC
llvm: 2.5+ million LOC

If debugging is the process of removing bugs, then programming must be the process of putting them in.
— Edsger Dijkstra

There are two ways to write error-free programs; only the third one works.
— Alan J. Perlis

My dear Compiler: What did you do there?

What does that mean?

- Test the code you deploy!
- Do not deploy the code without the debugging options used!

My dear Compiler: What did you do there?

What does that mean?

- Test the code you deploy!
- Do not deploy the code without the debugging options used!
- New hardware (revision) $\Rightarrow$ new tests!

My dear Compiler: What did you do there?

What does that mean?

- Test the code you deploy!
- Do not deploy the code without the debugging options used!
- New hardware (revision) $\Rightarrow$ new tests!

Is hardware bug-free?

My dear Compiler: What did you do there?

What does that mean?

- Test the code you deploy!
- Do not deploy the code without the debugging options used!
- New hardware (revision) $\Rightarrow$ new tests!

Is hardware bug-free?

- Pentium FDIV bug (1994)
- Phenom TLB bug (2007, detected pre-release)
- ATmega1280 Manual, Chapter 13: Errata
- ...

Keep Things Simple

Action on absolute sum

Implement a function which takes three signed 8-bit values and a function pointer. If the absolute sum of the three values is greater than 100, call the function defined by the function pointer.

Keep Things Simple

Action on absolute sum

Implement a function which takes three signed 8-bit values and a function pointer. If the absolute sum of the three values is greater than 100, call the function defined by the function pointer.

We start with the function definition:

```
void action_on_absolute_sum(void (*action)(void), int8_t a, int8_t b, int8_t c);
```

Keep Things Simple

First, we ensure that the three values are positive:

Keep Things Simple

First, we ensure that the three values are positive:

```
if (a < 0) {
  a = -a;
}

if (b < 0) {
  b = -b;
}

if (c < 0) {
  c = -c;
}
```

Keep Things Simple

Can we make it better?

- This is DRY (Don't repeat yourself, c&p, ...)
- Same functionality as `abs(3)` ?!

Keep Things Simple

Can we make it better?

- This is DRY (Don't repeat yourself, c&p, ...)
- Same functionality as `abs(3)` ?!

Let's use `abs(3)`: its signature is `int abs (int __i)`, thus we up-cast, stay in the same value range, and then down-cast. That fits!

Keep Things Simple

Can we make it better?

- This is DRY (Don't repeat yourself, c&p, ...)
- Same functionality as `abs(3)` ?!

Let's use `abs(3)`: its signature is `int abs (int __i)`, thus we up-cast, stay in the same value range, and then down-cast. That fits!

```
a = abs(a);
b = abs(b);
c = abs(c);
```

Keep Things Simple

Does it?

Let's use `abs(3)`: its signature is `int abs (int __i)`, thus we up-cast, stay in the same value range and then down-cast.

Keep Things Simple

Does it?

Let's use `abs(3)`: its signature is `int abs (int __i)`, thus we up-cast, stay in the same value range and then down-cast.

`INT8_MAX = 0x7f = 127` and `INT8_MIN = (-INT8_MAX - 1) = -128`.
`-INT8_MIN` is not in `int8_t`!
Thus, `abs(INT8_MIN) = INT8_MIN` even though `INT8_MIN` < 0!

Keep Things Simple

Does it?

Let's use `abs(3)`: its signature is `int abs (int __i)`, thus we up-cast, stay in the same value range and then down-cast.

`INT8_MAX = 0x7f = 127` and `INT8_MIN = (-INT8_MAX - 1) = -128`.
`-INT8_MIN` is not in `int8_t`!
Thus, `abs(INT8_MIN) = INT8_MIN` even though `INT8_MIN` < 0!

NB: This is not an issue with `INT8_MIN`!

> *Trying to take the absolute value of the most negative integer is not defined.*
> — *abs(3)*

Keep Things Simple

How can we fix this?

1. cheap out: Change the definition of `action_on_absolute_sum` to be undefined if one of the three values is `INT8_MIN`.
2. Define that `INT8_MIN` will be treated as `INT8_MIN+1`.
3. Handle it.

Keep Things Simple

How can we fix this?

1. cheap out: Change the definition of `action_on_absolute_sum` to be undefined if one of the three values is `INT8_MIN`.
2. Define that `INT8_MIN` will be treated as `INT8_MIN+1`.
3. Handle it.

Unless it is guaranteed that the value range of the three values does not include `INT8_MIN`, options 1 and 2 can be highly dangerous!

Keep Things Simple

How can we fix this?

1. cheap out: Change the definition of `action_on_absolute_sum` to be undefined if one of the three values is `INT8_MIN`.
2. Define that `INT8_MIN` will be treated as `INT8_MIN+1`.
3. Handle it.

Unless it is guaranteed that the value range of the three values does not include `INT8_MIN`, options 1 and 2 can be highly dangerous!

How to handle it?

In this case, it is quite simple: We perform a manual absolute conversion, and in case of an overflow we increment a counter. This counter is later considered in the sum.

Keep Things Simple

```
int8_t absForSum(int8_t val, uint8_t *carry)
{
  if (val == INT8_MIN) {
    (*carry)++;
    return INT8_MAX;
  }
  else {
    return abs(val);
  }
}

...

uint8_t sum = 0;
a = absForSum(a, &sum);
b = absForSum(b, &sum);
c = absForSum(c, &sum);
```

Keep Things Simple

Finishing it up

We now sum up the three variables and sum (carry). If the result is greater than 100, we call the function pointer.

```
sum += a + b + c;

if (sum > 100) {
  action();
}
```

Keep Things Simple

Not so fast!

The sum can get larger than `UINT8_MAX`!
Again, we have to deal with this.

- Use a larger type for the sum.

Keep Things Simple

Not so fast!

The sum can get larger than `UINT8_MAX`!
Again, we have to deal with this.

- Use a larger type for the sum.
- Newer compiler version have so called builtin-functions (e.g., `__builtin_add_overflow`), which can be used to detected an overflow and saturate the result.

Keep Things Simple

Not so fast!

The sum can get larger than `UINT8_MAX`!
Again, we have to deal with this.

- Use a larger type for the sum.
- Newer compiler version have so called builtin-functions (e.g., `__builtin_add_overflow`), which can be used to detected an overflow and saturate the result.
- Implement checks.

Keep Things Simple

Not so fast!

The sum can get larger than `UINT8_MAX`!
Again, we have to deal with this.

- Use a larger type for the sum.
- Newer compiler version have so called builtin-functions (e.g., `__builtin_add_overflow`), which can be used to detected an overflow and saturate the result.
- Implement checks.
  - The `SREG` on the AVR has flags which allow to detect overflows.

Keep Things Simple

Not so fast!

The sum can get larger than `UINT8_MAX`!
Again, we have to deal with this.

- Use a larger type for the sum.
- Newer compiler version have so called builtin-functions (e.g., `__builtin_add_overflow`), which can be used to detected an overflow and saturate the result.
- Implement checks.
  - The `SREG` on the AVR has flags which allow to detect overflows.
  - Sanity checks after the operations.
    Attention: In ISO-C the behavior of a signed overflow is undefined. Checkout the compiler flags `-fwrapv` and `-fstrict-overflow` (enabled with `-O2`, `-O3`, and `-Os`).

Summary

- C is convenient.
- The C compiler produces assembler code.
- Analysis of the passage from C code to assembler is exemplary for passage from a high-level language (or other design instrument) to a lower-level one.
- This lecture introduced you to the analysis of compiled C programs.
- Now it's your turn:
  get acquainted with GCC and start trying things out!

Questions?