EE 675: Microprocessor Applications in Power Electronics

M.C. Chandorkar
Department of Electrical Engineering
Indian Institute of Technology – Bombay

EE 675 Course Content

- Review of digital processors
- Fixed- and floating-point processors and microcontrollers
- Number formats and operations
- Assemblers and assembly language programming
- Binary file formats
- Introduction to power electronics control applications
- Implementing power electronics control on digital systems
- Numerical integration
- Real-time operating systems in power electronics control
- Hardware-in-loop simulation concepts

Reference: K. Ogata, “Discrete Time Control Systems”, second edition, Pearson International

Logistics

- Evaluation
  - Assignments: 40%
  - Mid-semester exam: 20%
  - End-semester exam: 40%
- Exams will be held according to the Academic Office timetable
- Assignments to be done in groups of three
  - One TMS320VC33 DSP board issued per batch

EE 675: Microprocessor Applications in Power Electronics

Review of Digital Processors

Microcontrollers

- Examples: Motorola 68HC16, Siemens SAB 88C166
- One operation per instruction
- Single instruction per cycle, low clock speed
- Several instructions to perform a given operation
- Narrow instruction word (8-bit or 16-bit)
- Multiple cycles for multiply
- Rich in on-chip peripherals, good interrupt structure
- Low cost

Microprocessors

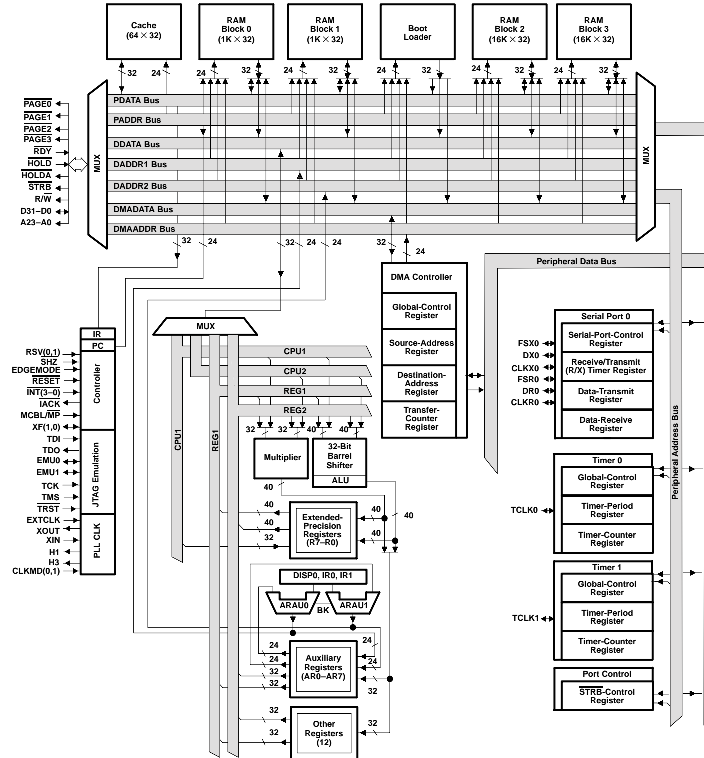
- Examples: Intel Pentium, Motorola Power PC
- Typically, one operation per instruction
- Multiple instructions per cycle
- Several instructions to perform a given operation
- Large program memory requirement
- Very good high-level software support (C compilers etc.)
- Expensive

Digital Signal Processors

- Examples: TMS320F240, TMS320F2407, TMS320VC33, DSP56300, ADSP2100
- Often, multiple instructions per clock cycle
- 24- and 32-bit processors have multiple instructions in one word
- Single instruction may be sufficient to perform a given operation
- CPU is designed for computation-intensive jobs (fixed / floating point)
- Relatively small program memory requirement
- On-chip bootloader; instruction cache
- Relatively low to medium cost

TMS320VC33 DSP Schematic Diagram

functional block diagram



TMS320VC33 Floating Point DSP Summary

- 32-bit floating point DSP with 13 ns instruction cycle
- Good for computation intensive embedded control applications
- On-chip PLL for 75 MHz internal clock generation
- 34K x 32-bit on-chip RAM
- On-chip program bootloader
- 32-bit instruction word
- 24-bit address space
- Two 32-bit timers
- Eight 40-bit extended precision registers
- Parallel instructions

EE 675: Microprocessor Applications in Power Electronics

Applications, Introduction to number representations

DSP Microcontrollers

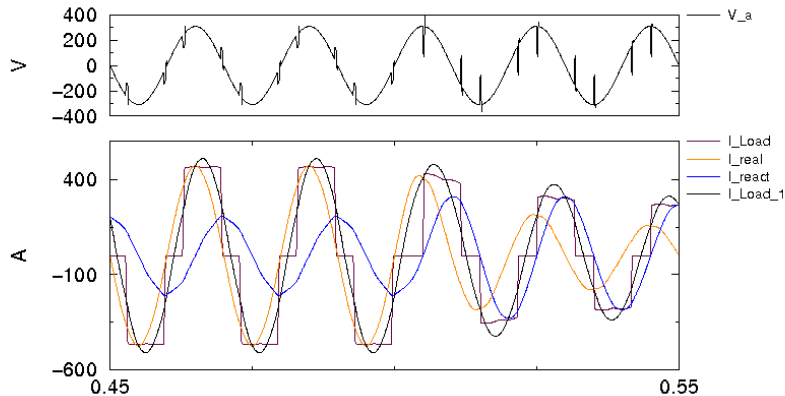
- Examples: TMS320F240, TMS320F243, TMS320F2407, DSP56800
- Combination of DSP and Microcontroller
- Often, multiple instructions per clock cycle
- Typically, 16 bit data and instructions
- CPU is designed for computation-intensive jobs
- Rich set of on-chip peripherals (ADCs, I/O ports, Flash memory etc.)
- 5 V as well as 3.3 V processors are on the market
- Relatively low to medium cost
- Reasonably good set of software development tools

TMS320F240 DSP Microcontroller Summary

- 16-bit fixed point DSP with 50 ns instruction cycle
- Good for closed loop control applications
- On-chip PLL for 20 MHz internal clock generation
- 16 k x 16 internal Flash Memory Module
- 2 x 8 analog input channels to 10 bit ADC
- Three 16-bit general purpose timers
- Very rich interrupt structure
- PWM generation
- 132 pin PQFP package

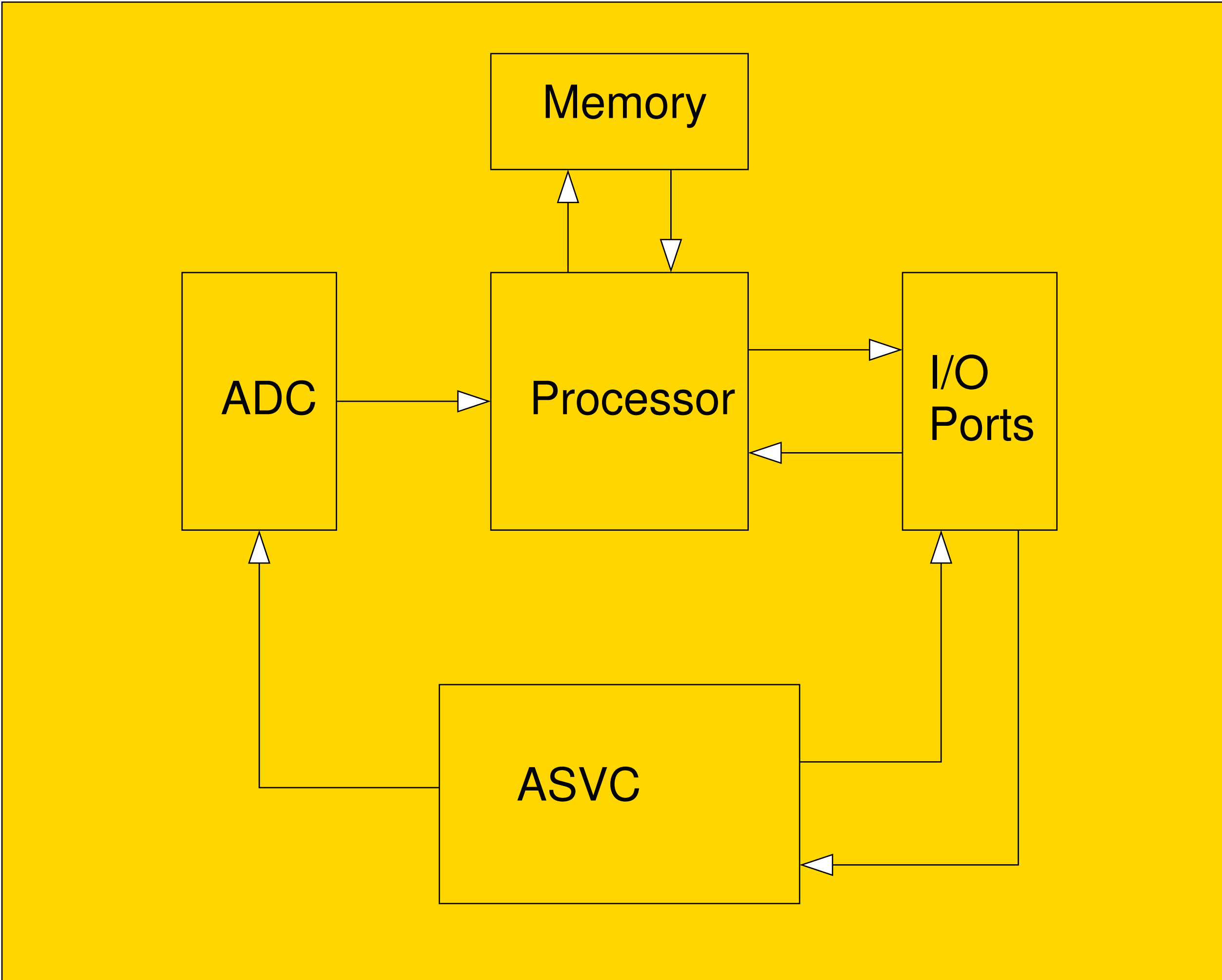
Advanced Static VAr Compensation

as A bs A cs A c b a rla rlb rlc a1 b1 c1 np ap bp cp n ASVC Load

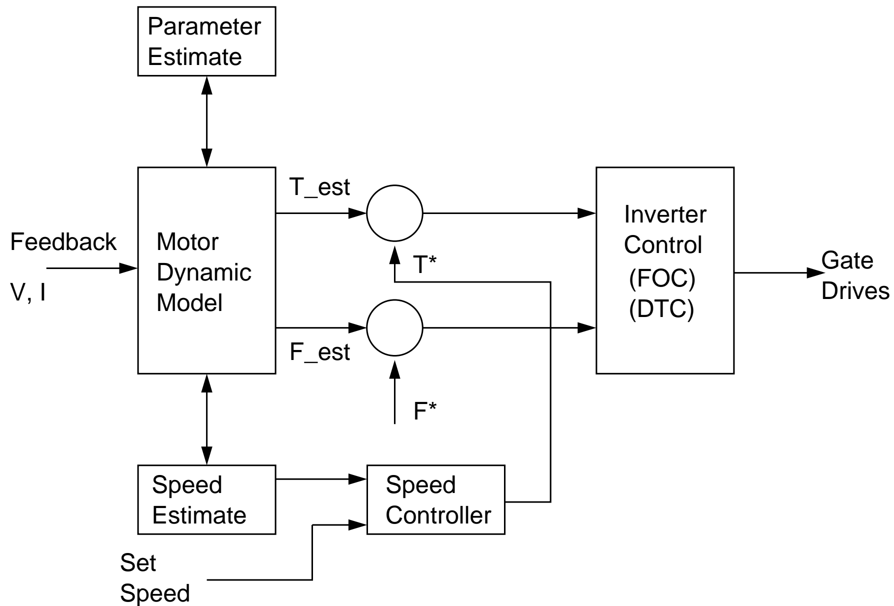


- ASVC: Advanced Static VAr Compensator
- Provides all of the reactive current component
- Provides harmonic currents upto the 17th
- Good PLL design needed

Schematic Layout of Digital Processor Control Hardware



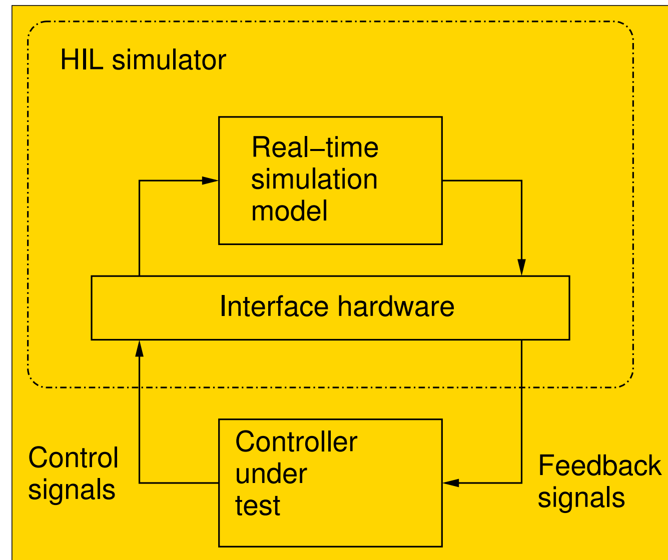
High Performance Drive Control



Time Frames

- 20 μs:
  - Motor model
  - Feedback
- 100 μs:
  - Motor model
  - Protection
  - Speed estimate
- 1 ms:
  - Flux reference
  - Speed controller
  - Parameter estimates
  - Ride through

Hardware-In-Loop Simulation



- Useful in the testing and tuning of digital control hardware and software
- Reduces development time and costs
- Avoids potentially hazardous situations in controller testing
- *Example:* Testing and setting of power system protection relays
  - Cannot induce faults on a live power system to tune the relay
  - Can simulate faults on a HIL simulator to tune the relay

EE 675: Microprocessor Applications in Power Electronics

Number representations

Number Representations on Fixed Point Processors

- ADC outputs need to be represented properly in a processor
- Control system variables need to be represented properly
- Typically 16, 24 or 32 bit numbers
- Need to represent positive and negative numbers
- Signed integer and signed fraction representations are the most common

Sixteen-bit Signed Integers

- The 2's complement notation is used to represent signed integers
- Positive numbers: Represent directly as binary numbers
- Negative numbers: Take the simple complement and add 1

Example: +3

`0000 0000 0000 0011`

Example: -3

`0000 0000 0000 0011` $\rightarrow$ **`1111 1111 1111 1100`**

$\rightarrow$ **`1111 1111 1111 1101`**

- Negative numbers start with an MSB of 1
- Positive numbers start with an MSB of 0
- Number range:
  - Most positive number is $+2^{15} - 1 = +32767$
  - Most negative number is $-2^{15} = -32768$
  - Number range: 65535

Operations with 16-bit 2's Complement Numbers

- Addition: Binary addition

```
0000 0000 0000 0011 +
0000 0000 0000 0011 =
0000 0000 0000 0110
```

- Subtraction: Binary addition with 2's complement (The carry bit is discarded)

```
0000 0000 0000 0011 +
1111 1111 1111 1101 =
0000 0000 0000 0000
```

Operations with 16-bit 2's Complement Numbers

- Multiplication: The result is a 32-bit signed integer

  0x0003 $\times$ **0xfffd** $=$ **0xfffffff7**

- The results of multiplications cannot always be stored in the 16-bit number range

  0x012c $\times$ **0x012c** $=$ **0x00015f90**
  (300 $\times$ **300** $=$ **90000)**

Signed Fraction Representation with 16-bit Numbers

- Use the 16 bits to represent signed fractions in the range -1 to 1
- An imaginary decimal point is placed between bit 15 and bit 14. MSB indicates the sign and the remaining 15 bits represent the fraction
- This is the Q-15 fractional format

Signed Fraction Representation with 16-bit Numbers

- Signed decimal fraction to Q-15 conversion:
  - Multiply the fraction with the scaling factor $2^{15} = 32768$
  - Convert the resulting integer to hexadecimal
  - If the fraction is negative, take the 2's complement
- *Example:*

  +0.25: **+0.25** $\times$ **32768** $=$ **08192** $\rightarrow$ **0x2000**

  -0.50: **+0.50** $\times$ **32768** $=$ **16384** $\rightarrow$ **0x4000** $\rightarrow$ **0xc000**

- The resolution of the fractional numbers is $2^{-15} \approx 0.00003051757$
- To convert a Q-15 number to its fractional value, reverse the procedure given above

Operations with Q-15 Numbers

- Addition: Hex addition
  0x2000 $+$ **0x2000** $=$ **0x4000**
  (0.25 $+$ **0.25** $=$ **0.50)**
- Subtraction: Hex addition with 2's complement
  0x2000 $+$ **0xc000** $=$ **0xe000**
  (0.25 $-$ **0.50** $=$ $-$**0.25)**

EE 675: Microprocessor Applications in Power Electronics

Number representations, Assembler operations

Operations with Q-15 Numbers

- Need to be careful of overflows and underflows

  $\texttt{0x6000} + \texttt{0x6000} = \texttt{0xc000}$

  $(\texttt{0.75} + \texttt{0.75} = -\texttt{0.5 ???})$

- Many processors have an "overflow mode" which prevents the result of an addition or subtraction from going beyond 0x7fff and 0x8000 respectively
- Overflows and underflows can be handled by software
- But it is better to avoid them in the first place

  $\texttt{0.1} \times (\texttt{0.5} + \texttt{0.7}) = \texttt{0.05} + \texttt{0.07}$

  and not

  $\texttt{0.1} \times \texttt{1.2}$

Operations with Q-15 Numbers – cont'd

- Multiplication:

  The multiplication of two Q-15 numbers results in a Q-30 number. Consider two decimal fractions x and y. Their multiplication z' is

$$z' = (x \times 2^{15}) \times (y \times 2^{15}) = xy \times 2^{30},$$

which is the Q-30 representation of the desired fractional result $z = xy$.

To get the Q-15 representation of the result, it is necessary to multiply z' as follows.

$$z \times 2^{15} = z' \times 2^{-15}$$

That is, z' needs to be shifted right by 15 bits. Equivalently, we may shift z' left by one bit, and take the most significant 16 bits of the resulting number to be the Q-15 representation of the result z; this is the preferred method.

Operations with Q-15 Numbers – cont'd

Formula: Multiply two Q-15 numbers, shift the 32-bit result left by one bit and take the most significant 16 bits to get the Q-15 result

Example:

0x4000 $\times$ **0x6000** $=$ **0x18000000** $\rightarrow$ **0x30000000** $\rightarrow$ **0x3000**

(0.5 $\times$ **0.75** $=$ **0.375)**

EE 675: Microprocessor Applications in Power Electronics

Floating-point representations, Assembler operations

Floating Point Number Representation

e: 8-bit exponent field, contains signed 2's complement integer
s: Sign bit
f: Fractional part of the mantissa
x: Floating point number

- If $s = 0$: $x = 01.f \times 2^e$
- If $s = 1$: $x = 10.f \times 2^e$
- *Example:* $+1.7 \times 10^1$

  $+17 = 01.f \times 2^e$, $s = 0$

  $+17 = 01.0625 \times 16$

  $e = 4$ and $f = 00010000000000000000000$

$\texttt{x} =$ **`0000 0100 0000 1000 0000 0000 0000 0000`**
$=$ **`0x04080000`**

32-bit Floating Point Number Range

- Most positive number:

  0x7f7fffff $=$ $+$**3.4028234**$\times$ **10**^{38}

- Least positive number:

  0x81000000 $=$ $+$**5.8774717**$\times$ **10**$^{-39}$

- Most negative number:

  0x7f800000 $=$ $-$**3.4028236**$\times$ **10**^{38}

- Least negative number:

  0x81ffffff $=$ $-$**5.8774724**$\times$ **10**$^{-39}$

The Tasks of an Assembler

- Convert assembly language code into machine binary code.

  Example:

```
mpy #254 ⇝⇝ 1100000011111110
```

- Handle symbolic information.

  Example:

```
Start:  mpy   #254
        bcnd  Start, LT
        add   #1
        bcnd  End, GT
        add   #1
End:    b     Start
```

- Handle expressions in operands.

  Example:

```
b  (Label + (0x10 * 1011b))
```

- Handle sections.

  Example:

```
.sect  ``vectors''
```

The Tasks of an Assembler - continued ...

- Handle external references.

  Example:

```
.ref  ext1 ; ext1 is defined in another file
...
...
add   ext1 ; ext1 used in this file, unknown value
```

- Handle macro substitution.

  Example:

```
S2E  .macro  qs,ds,cos,sin,qe,de
     lt      qs
     mpy     cos
     ltp     ds
     mpy     sin
     mpya    cos
     sach    qe
     ltp     qs
     mpy     sin
     spac
     .endm
```

The Tasks of an Assembler - continued ...

- Handle relocation information.

  Example:

  ```
  b  Label ; A relocation entry is created
  ```

- Report errors.

  Example:

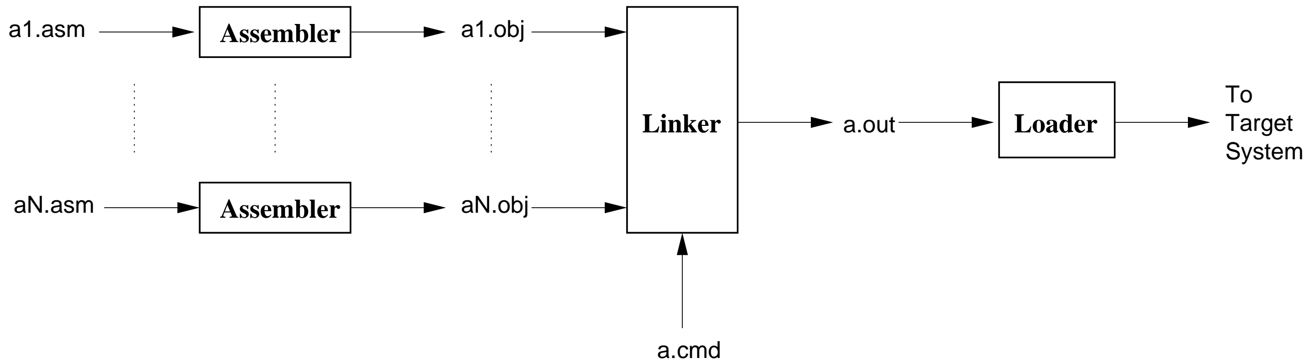
  ```
  add  %!*#$@ ; Causes an error message
  ```

- An assembler usually creates an object file (extension *.obj*), which contains unresolved externals, and internal symbol information for other object modules.
- An object file has relocation entries for each section.

The Tasks of an Assembler - concluded

- Texas Instruments DSP software tools usually create object files in the *Common Object File Format* (COFF).
- A *linker* connects all the object files by:
  - joining up sections from different files,
  - resolving all symbolic references,
  - performing relocations,
  - creating absolute code.
- The final linked code file has no unresolved symbolic references, and no relocation information.

The Code Development Process



- The entire application is usually spread among several source files.
- The linker links all object files to make the final executable file.
- The *.cmd* file contains the memory map of the target system.
- The *.obj* and the *.out* files often have the same file format. These include:
  - Common Object File Format (COFF),
  - Executable and Linking Format (ELF).

Sections

- A *section* is the basic structural unit of assembly language code.
- A section typically contains program or data code.

    Example:

```
Start  .text  ; Program code section
       b      Init_DSP
       ...
       .data  ; Initialised data section
       .word  0x7fff, 08000h, 1011001b
       ...
```

- Usually there are three default sections: *.text*, *.data* and *.bss*.
- Named sections can also be defined by users.

    Example:

```
      .sect  ``vect_table''
Int0  b      INT0
Int1  b      INT1
      ...
```

- The linker combines section code across different files.

EE 675: Microprocessor Applications in Power Electronics

Assembler operations, TMS320VC33 details

COFF File Structure

- File headers
- Section headers
- Section raw data
- Section relocation information
- Section line numbers
- File symbol table
- String table to hold long symbol names
- All TI COFF versions have the same format but different structure sizes.
- TI COFF comes in three versions: COFF0, COFF1 and COFF2.
- COFF was used in the Unix System V.

file header
optional file header
section 1 header
section n header
section 1 raw data
section n raw data
section 1 reloc info
section n reloc info
section 1 line nos.
section n line nos.
symbol table
string table

File Header

- COFF 0: 20 bytes long ; COFF 1 and COFF 2: 22 bytes long
- Target DSP ID
- Number of section headers
- Time and date stamp of file creation
- Symbol table file pointer
- Number of entries in the symbol table
- Flags
  - File executable or not (unresolved references?)
  - File has reloc information or not
  - Byte ordering:

    Little Endian - Object data LSB first

    Big Endian - Object data MSB first

Section Headers

- COFF 0 and COFF 1: 40 bytes long ; COFF 2: 48 bytes long
- Section name and its address in the system memory
- File pointer to section raw data
- File pointer to the section s relocation entries
- Number of relocation entries in the section
- By default, section headers are always created for the .text, .data and .bss sections
- Section names for all user defined sections created with the .sect directive are significant to 8 characters

Section Relocation Entries

- A relocation entry is created when the assembler encounters a symbol used as an argument of an assembly language instruction.

  Example:

```
         .global  Label2
         .text
         ...
         b        Label1 ; reloc entry generated
         ...
         ...
Label1   nop
         and      Data1 ; reloc entry generated
         b        Label2 ; reloc entry generated
         .data
Data1    .word    0123h
```

- The reloc entry specifies the virtual address, the symbol table index, and the reloc type (7 LSBs of address, 9MSBs of address, 16 bit address, ...).

The COFF Symbol Table

- The symbol table appears at the end of the COFF file, followed by the string table.
- Each symbol table entry is 18 bytes long.
- Each entry contains information on:
  - The symbol name.
  - The symbol value.
  - The section ID of the symbol.
  - The symbol type (integer, float, double, character, pointer, array, ...)
  - The symbol storage class (static, automatic, external, label, ...)
- Symbol names can have any length. If a name is longer than eight characters, then the symbol name entry contains a pointer into the String Table, which contains the characters that make up the symbol name.
- By default, all section names and global symbols are entered into the COFF symbol table. Local symbols are not entered.

Assembler Passes

- An assembler makes two passes.
- In Pass 1, it creates an internal symbol table.
  - Symbols specified with a `.global`, `.def` or `.ref` directive are entered into this table.
  - All labels are also entered into the internal symbol table. The value given to each label is the value of the *section program counter* at that point.
  - It calculates the lengths of various sections.
  - It calculates the values of the file pointers.
- In Pass 2, the assembler actually generates assembly code.
  - It recognises opcodes and operands.
  - It retrieves symbol values and evaluates expressions.
  - It forms the machine binary code and writes it to the output COFF file.
  - It writes the reloc entries, the symbol table and the string table to the COFF file.

Assembling Process: Example

```
      .text
      clrc INTM                  ; SPCt = 0
      addc *, ar0                ; SPCt = 1
      bcnd L2+(2<<1), LT         ; SPCt = 2
      mac L2, 2                  ; SPCt = 4
      lacc L3, 9                 ; SPCt = 6

      .data
x     .word 0xa, 0eh             ; SPCd = 0
      .word 10100b               ; SPCd = 2
      .word x                    ; SPCd = 3

      .text                      ; SPCt = 6
L2:   neg                        ; SPCt = 7
L3:   cmpl                       ; SPCt = 8

      .data                      ; SPCd = 3
y     .word 0x4                  ; SPCd = 4
```

TMS320VC33 Description

- Memory map
- Interrupts
- Addressing modes

TMS320VC33 Interrupt Processing

- Five interrupts from internal peripherals
- Four external interrupts from pins

Microcomputer mode:

Name	Address	Priority	Function
INT0	0x809fc1	1	External on pin /INT0
INT1	0x809fc2	2	External on pin /INT1
INT2	0x809fc3	3	External on pin /INT2
INT3	0x809fc4	4	External on pin /INT3
XINT0	0x809fc5	5	Serial port XMT buffer empty
RINT0	0x809fc6	6	Serial port RCV buffer full
TINT0	0x809fc9	7	Timer0 interrupt
TINT1	0x809fca	8	Timer1 interrupt
DINT	0x809fcb	9	DMA controller interrupt

TMS320VC33 Interrupt Behaviour

- Three CPU registers associated with interrupt processing.
  1. Status register (ST)
  2. Interrupt Enable register (IE)
  3. Interrupt Flag Register (IF)
- The Global Interrupt Enable (GIE) bit in the ST has to be set for interrupts to be enabled (upon CPU reset it is cleared)
- The individual interrupt enable bit has to be set in the IE (interrupts are disabled upon CPU reset)
- When an interrupt is recognised, the corresponding IF bit is set
- The IF bits may be polled or written to by user programs

Interrupt Handling

Upon recognising an interrupt the CPU does the following in sequence

1. Disable interrupts (0 $\rightarrow$ GIE)
2. Set the proper IF bit
3. PC $\rightarrow$ *(++SP)
4. (Interrupt vector) $\rightarrow$ PC

- The user program must re-enable interrupts
- If the interrupt service routine (ISR) needs to be interruptible, interrupts may be re-enabled upon entry into the ISR

TMS320VC33 Addressing Modes

- How are operands of instructions addressed?
- Operand addresses are 24 bits long
- There are five main addressing modes:
  1. Register addressing: A CPU register contains the operand
  2. Direct addressing: A part of the operand's address is contained in the instruction
  3. Indirect addressing: An auxiliary register contains the operand's address
  4. Immediate addressing: The operand is contained within the instruction
  5. PC-relative addressing: The instruction contains a displacement to the program counter

Addressing Modes

- Register addressing: A CPU register contains the operand
  - *Example:*

```
absi  R0
```

- Direct addressing:

  The Data Page Pointer (DP) contains $A_{23} - A_{16}$

  The instruction contains $A_{15} - A_0$

  The processor concatenates the DP and the instruction fields
  - *Example:*

```
ldp  @cos
ldf  @cos, R0
```

Addressing Modes

- Indirect addressing: An auxiliary register contains the operand address
- AR modification is possible in the same instruction
  - *Example:*

```
stf  R0, *+AR7(1)   ; address = AR7 + 1
stf  R0, *-AR7(1)   ; address = AR7 - 1
stf  R0, *++AR7(2)  ; address = AR7 + 2, AR7 = AR7 + 2
stf  R0, *--AR7(2)  ; address = AR7 - 2, AR7 = AR7 - 2
stf  R0, *AR7++(3)  ; address = AR7, AR7 = AR7 + 3
stf  R0, *AR7--(3)  ; address = AR7, AR7 = AR7 - 3
```

- All of the above are also possible using a displacement contained in the Index Registers IR0 and IR1
  - *Example:*

```
stf  R0, *+AR7(IR0)   ; address = AR7 + IR0
stf  R0, *AR7--(IR1)  ; address = AR7, AR7 = AR7 - IR1
```

Addressing Modes

- Immediate addressing: The operand is contained in the instruction
- May be a 16-bit (short immediate format) operand
  - *Example:*

```
addi  1, R0
mpyf  2.4e-2, R0
```

- May be a 24-bit (long immediate format) operand
  - *Example:*

```
call  0x800107
```

- PC-relative addressing: Used for branching. A signed 16-bit or 24-bit displacement is contained in the instruction. This is added to the PC.
  - *Example:*

```
label:  nop
        bz   label1 ; 16-bit displacement
        br   label1 ; 24-bit displacement
```

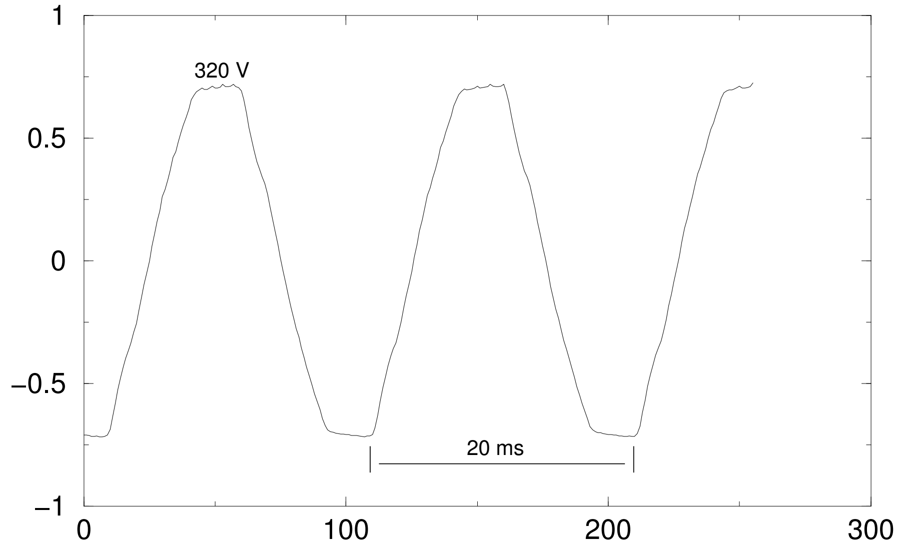
EE 675: Microprocessor Applications in Power Electronics

Using the TMS320VC33 Hardware and Code Generation Tools, Assignment 1

EE 675: Microprocessor Applications in Power Electronics

Power Electronic Control Applications – Compensation

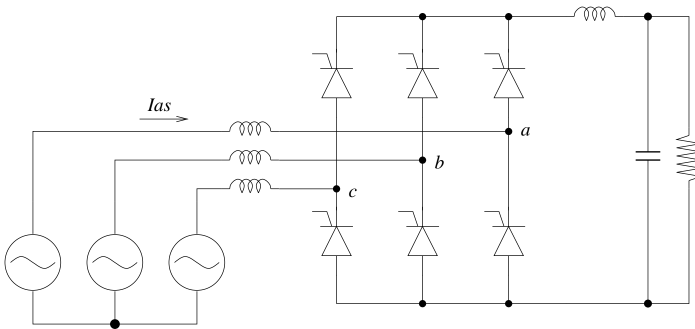
Measured AC Voltage Waveform

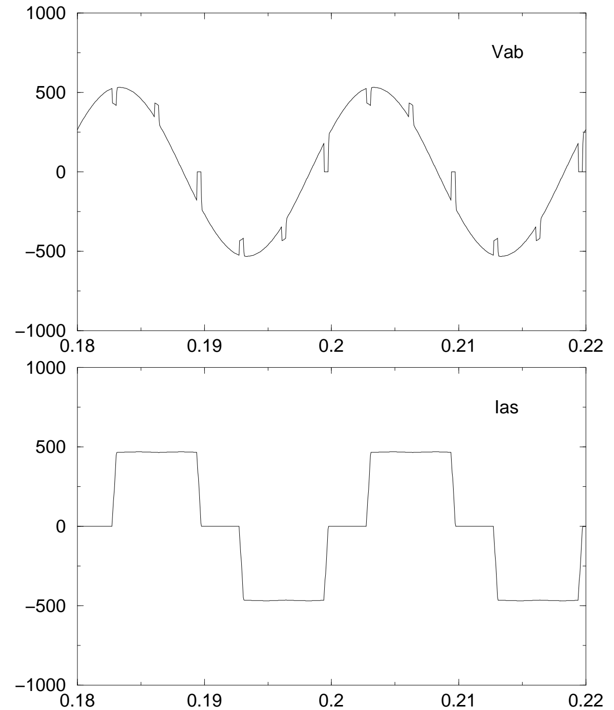


single–phase supply voltage: 01.04.2005 11:15

- Waveform recorded from a single-phase outlet on the IIT Bombay campus.
- Voltage distortion is caused by many computers connected to the mains.

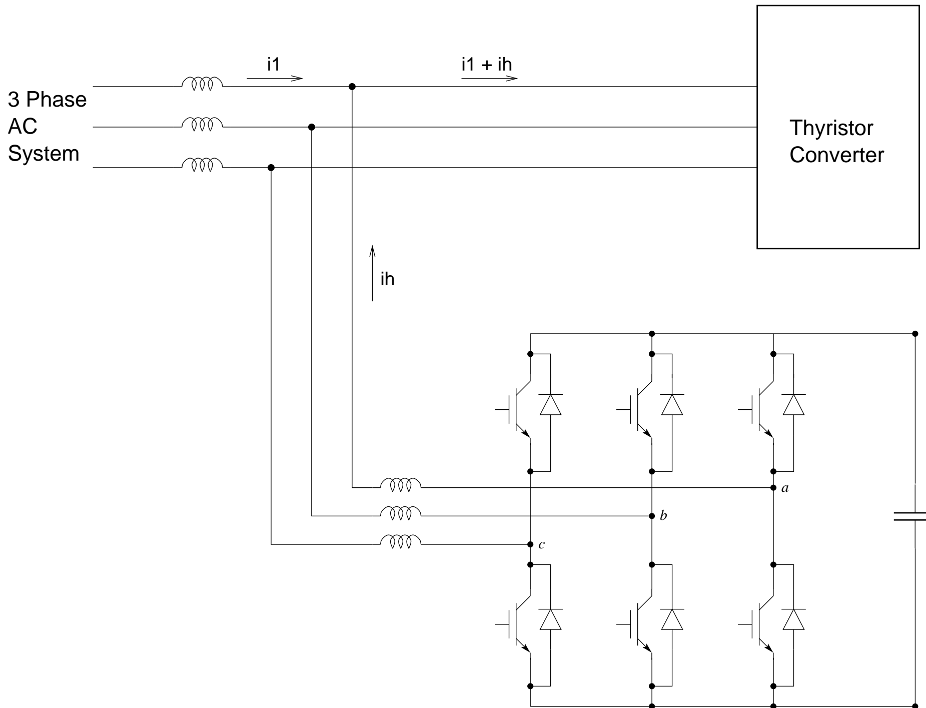
Non-sinusoidal Input Currents





- The line current is non-sinusoidal.
- The line-line voltage is notched.
- Problems for other loads connected to a, b, c terminals.

Mitigating the Harmonics Problem: Active Filtering



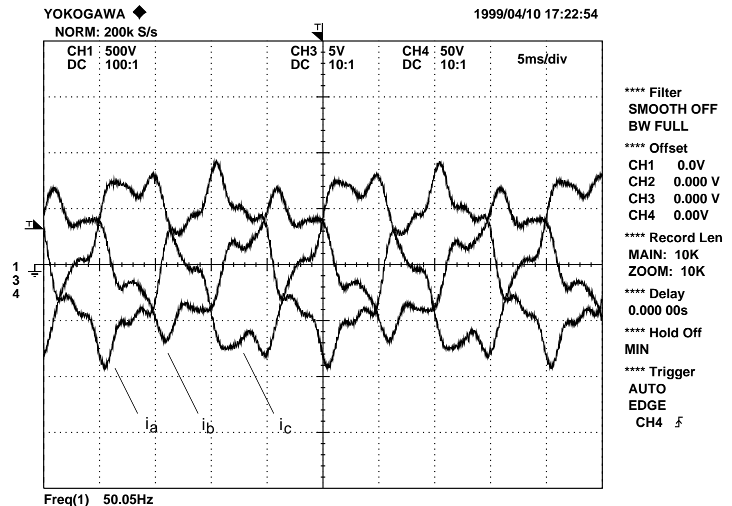
- The ac source provides only the fundamental component of the load current.
- The same circuit can be used for reactive power compensation.
- Extract the harmonic components i_h from the load current.

EE 675: Microprocessor Applications in Power Electronics

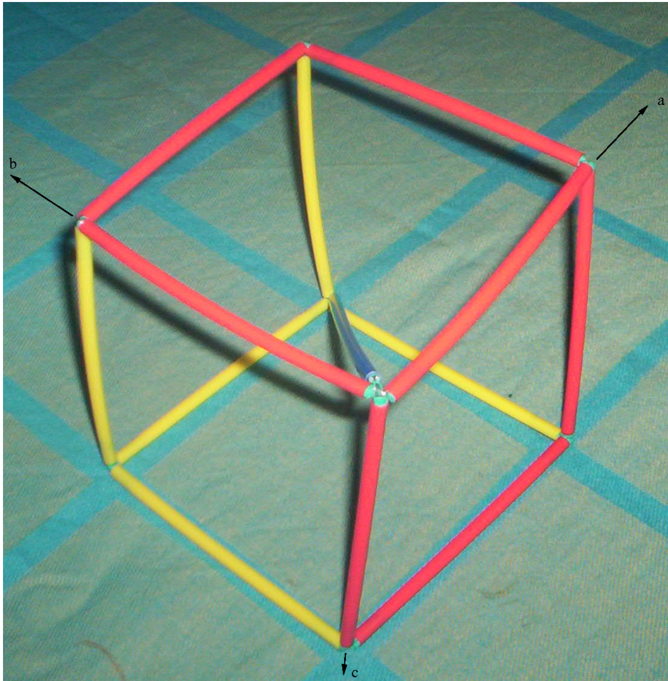
Reference Frame Transformations

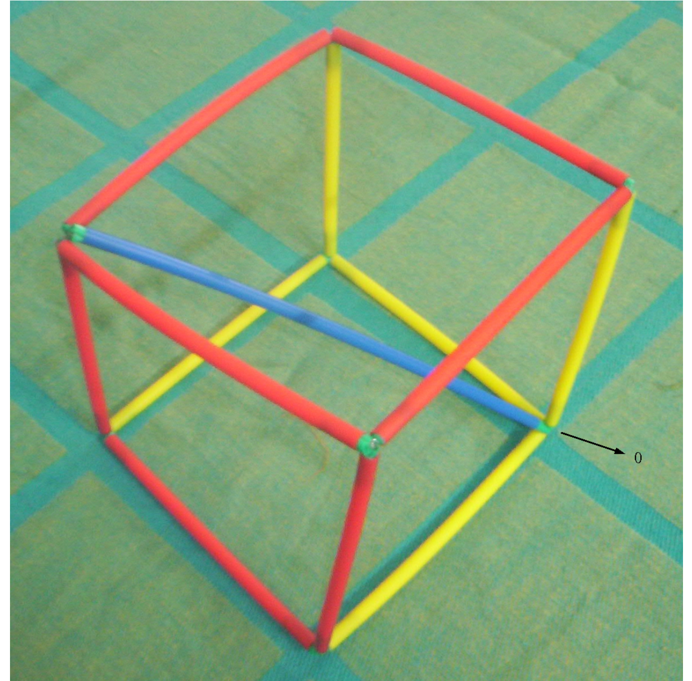
Unbalanced Non-sinusoidal Currents

- Input: Unbalanced three phase signals.
- The fundamental component as well as the harmonics are unbalanced.
- Example: Single phase traction supply derived from three phase distribution system.
- Task: To extract the sequence components for the fundamental and the harmonics.
- This task has to be done in real time on a DSP system.
- One way to do this is to use reference frame transformations.



Reference Frame Transformation





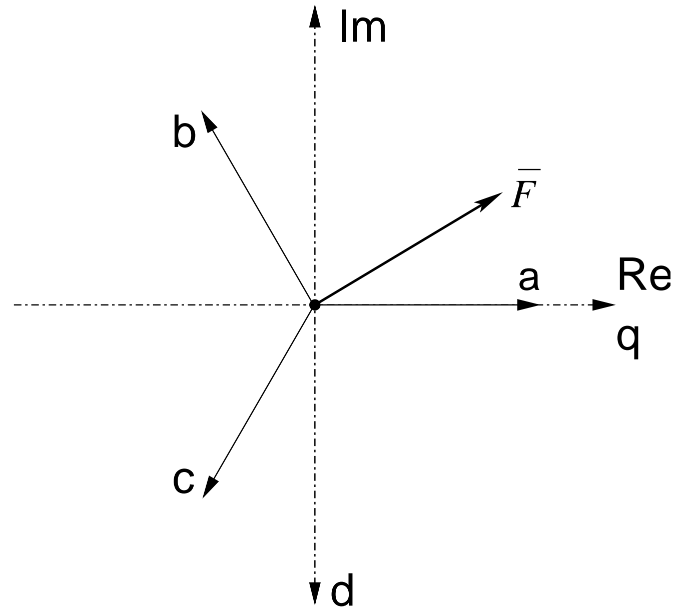
Transformation to a Stationary Reference Frame

$$\begin{pmatrix} f_q \\ f_d \\ f_0 \end{pmatrix} = \frac{2}{3} \begin{pmatrix} 1 & \cos\left(-\frac{2\pi}{3}\right) & \cos\left(+\frac{2\pi}{3}\right) \\ 0 & \sin\left(-\frac{2\pi}{3}\right) & \sin\left(+\frac{2\pi}{3}\right) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} f_a \\ f_b \\ f_c \end{pmatrix}$$

- f_a, f_b and f_c are the instantaneous a, b and c phase quantities.
- f_q, f_d and f_0 are the transformed quantities.
- We consider cases in which $f_a + f_b + f_c = 0$. In this case,

$$f_q = f_a$$
$$f_d = (f_c - f_b)/\sqrt{3}$$

- Complex number representation:
  $\bar{F} = f_q - jf_d$



Transformation Example

Measured:

$$\theta = \omega t$$
$$f_a = \cos(\theta)$$
$$f_b = \cos(\theta - 2\pi/3)$$
$$f_c = \cos(\theta + 2\pi/3)$$

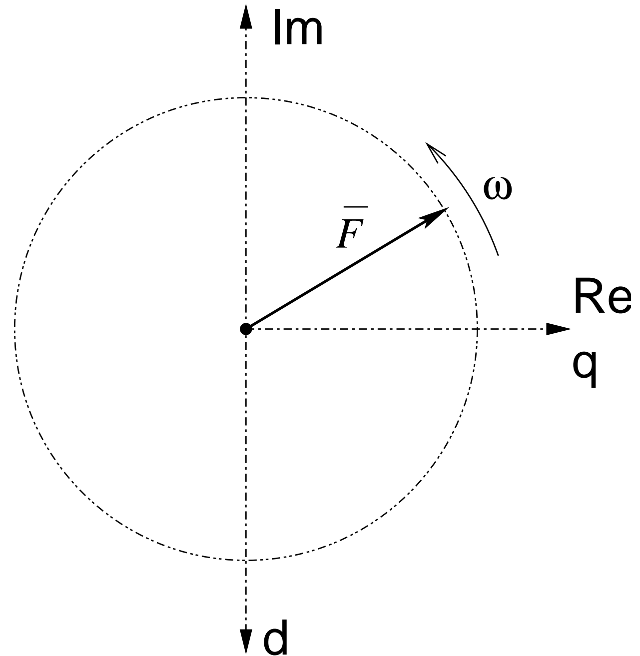
Transformed:

$$f_q = +\cos\theta$$
$$f_d = -\sin\theta$$

Complex number:

$$\bar{F} = \cos\theta + j\sin\theta$$
$$\bar{F} = e^{j\theta}$$

The vector $\bar{F}$ rotates anti-clockwise with an angular velocity of ω.



Transformation from abc to $dq0$

Transformation to a Rotating Reference Frame

Transformation:

$$\bar{F}^e = \bar{F} e^{-j\theta_e}$$
$$\theta_e = \omega_e t$$

Example:

$$\bar{F} = \cos\theta + j\sin\theta = e^{j\theta}$$

Transformed:

$$\bar{F}^e = e^{j(\theta-\theta_e)}$$

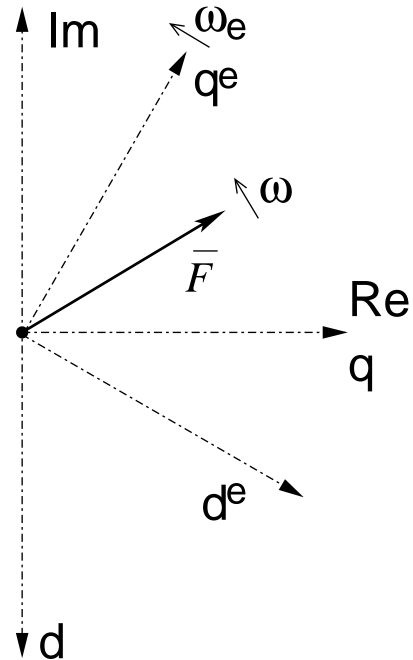
If $\theta = \theta_e$ for all time, then

$$\bar{F}^e = 1 + j0$$

$\bar{F}^e$ is a constant vector.

Transformation back to the stationary reference frame is achieved by

$$\bar{F} = \bar{F}^e e^{+j\theta_e}$$



Transformation equations:

$$f_q^e = f_q \cos\omega_e t - f_d \sin\omega_e t$$
$$f_d^e = f_q \sin\omega_e t + f_d \cos\omega_e t$$

Summary of Transformation Equations

- *a-b-c* to stationary *q-d*:

$$f_q = f_a$$
$$f_d = (f_c - f_b)/\sqrt{3}$$

- TMS320VC33 macro code:

```
isqrt3  .float 0.5773502
ABC2S   .macro a,b,c,qs,ds
ldf     a,r0
stf     r0,qs
ldf     c,r0
subf    b,r0
mpyf    @isqrt3,r0
stf     r0,ds
.endm
```

- Stationary *q-d* to *a-b-c*:

$$f_a = f_q$$
$$f_b = -\tfrac{1}{2} f_q - \tfrac{\sqrt{3}}{2} f_d$$
$$f_c = -\tfrac{1}{2} f_q + \tfrac{\sqrt{3}}{2} f_d$$

- Assumptions:

$$f_a + f_b + f_c = 0$$
$$\theta_e = \int \omega_e dt$$

Summary of Transformation Equations

- Stationary *q-d* to rotating *qe-de*:

$$f_q^e = f_q \cos\theta_e - f_d \sin\theta_e$$
$$f_d^e = f_q \sin\theta_e + f_d \cos\theta_e$$

- Rotating *qe-de* to stationary *q-d*:

$$f_q = +f_q^e \cos\theta_e + f_d^e \sin\theta_e$$
$$f_d = -f_q^e \sin\theta_e + f_d^e \cos\theta_e$$

TMS320VC33 Code for Stationary to Rotating Frame Transformation

```
S2EP  .macro qs,ds,cos,sin,qe,de
ldf    qs,r0    ;r0=qs
ldf    ds,r1    ;r1=ds
ldf    cos,r2   ;r2=cos
ldf    sin,r3   ;r3=sin
mpyf3  r2,r0,r4 ;r4=cos*qs
mpyf3  r3,r1,r5 ;r5=sin*ds
subf   r5,r4    ;r4=cos*qs-sin*ds
stf    r4,qe    ;store qe
mpyf3  r3,r0,r4 ;r4=sin*qs
mpyf3  r2,r1,r5 ;r5=cos*ds
addf   r5,r4    ;r4=sin*qs+cos*ds
stf    r4,de    ;store de
.endm
```

EE 675: Microprocessor Applications in Power Electronics

Harmonic and Unbalanced Current Extraction

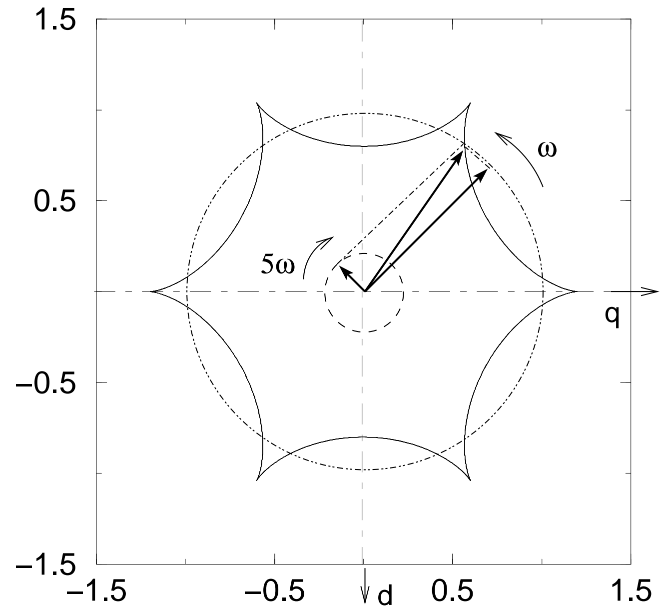
Example: Balanced Currents with Fifth Harmonic

$$
\begin{aligned}
i_a &= a_1 \cos \omega t + a_5 \cos 5\omega t \\
i_b &= a_1 \cos(\omega t - 2\pi/3) + a_5 \cos 5(\omega t - 2\pi/3) \\
i_c &= a_1 \cos(\omega t + 2\pi/3) + a_5 \cos 5(\omega t + 2\pi/3)
\end{aligned}
$$

- Stationary ref. frame transformation:

$$
\begin{aligned}
i_q &= +a_1 \cos \omega t + a_5 \cos 5\omega t \\
i_d &= -a_1 \sin \omega t + a_5 \sin 5\omega t \\
\bar{I} &= i_q - j i_d \\
&= a_1 e^{j\omega t} + a_5 e^{-j5\omega t}
\end{aligned}
$$

- The fundamental vector rotates anit-clockwise. The 5th harmonic vector rotates clockwise.
- The total current vector is the sum of the fundamental and 5th harmonic vectors.



Vector loci for $a_1 = 1$ and $a_5 = 1/5$, at $\omega t = \pi/4$.

Example: Unbalanced Currents

Measured currents:

$$i_a = \alpha \cos(\omega t)$$
$$i_b = \cos(\omega t - 2\pi/3)$$
$$i_c = -(i_a + i_b)$$

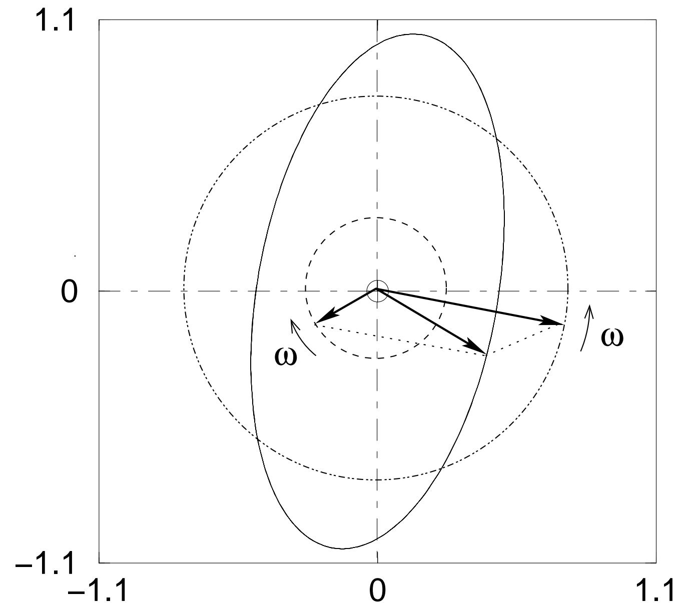
Transform to stationary reference frame:

$$i_q = \alpha \cos(\omega t)$$
$$i_d = \frac{(1-\alpha)}{\sqrt{3}} \cos(\omega t) - \sin(\omega t)$$
$$\bar{I} = i_q - j i_d$$

Vector representation:

$$\bar{I} = \bar{I}_+ e^{+j\omega t} + \bar{I}_- e^{-j\omega t}$$
$$\bar{I}_+ = \frac{(1+\alpha)}{2} - j\frac{(1-\alpha)}{2\sqrt{3}}$$
$$\bar{I}_- = -\frac{(1-\alpha)}{2} - j\frac{(1-\alpha)}{2\sqrt{3}}$$

The current vector $\bar{I}$ is composed of two components, $\bar{I}_+$ rotating anti-clockwise and $\bar{I}_-$ rotating clockwise, with the angular velocity ω.



Vector loci for $\alpha = 0.5$, at $t = 0$.

Extraction of Fundamental and Harmonic Components

- Example input:

$$
\begin{aligned}
i_a &= a_1 \cos \omega t + a_5 \cos 5\omega t \\
i_b &= a_1 \cos(\omega t - 2\pi/3) + a_5 \cos 5(\omega t - 2\pi/3) \\
i_c &= a_1 \cos(\omega t + 2\pi/3) + a_5 \cos 5(\omega t + 2\pi/3)
\end{aligned}
$$

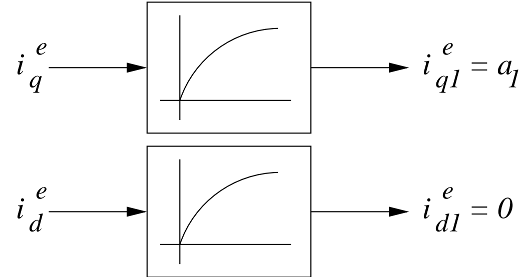
- We want to extract a_1.
- Stationary ref. frame transformation:

$$
\begin{aligned}
i_q &= +a_1 \cos \omega t + a_5 \cos 5\omega t \\
i_d &= -a_1 \sin \omega t + a_5 \sin 5\omega t \\
\bar{I} &= i_q - j i_d \\
&= a_1 e^{j\omega t} + a_5 e^{-j5\omega t}
\end{aligned}
$$

- Transformation to $+\omega$ ref. frame:

$$
\begin{aligned}
\bar{I}^e_{+1} &= \bar{I} e^{-j\omega t} = a_1 + a_5 e^{-j6\omega t} \\
i^e_q &= a_1 + a_5 \cos(6\omega t) \\
i^e_d &= a_5 \sin(6\omega t)
\end{aligned}
$$

- Low pass filter:



- Rotating $+\omega$ to stationary ref. frame:

$$
\begin{aligned}
i_{q1} &= +a_1 \cos(\omega t) \\
i_{d1} &= -a_1 \sin(\omega t)
\end{aligned}
$$

- Stationary ref. frame to abc:

$$
\begin{aligned}
i_{a1} &= a_1 \cos(\omega t) \\
i_{b1} &= a_1 \cos(\omega t - 2\pi/3) \\
i_{c1} &= a_1 \cos(\omega t + 2\pi/3)
\end{aligned}
$$

- The $5th$ harmonic component is extracted similarly by transforming and filtering in the -5ω rotating reference frame.

Extraction of Unbalance Components

- Example input:

$$
\begin{aligned}
i_a &= \alpha \cos(\omega t) \\
i_b &= \cos(\omega t - 2\pi/3) \\
i_c &= -(i_a + i_b)
\end{aligned}
$$

- Transform to stationary reference frame:

$$
\begin{aligned}
i_q &= \alpha \cos(\omega t) \\
i_d &= \tfrac{(1-\alpha)}{\sqrt{3}} \cos(\omega t) - \sin(\omega t) \\
\bar{I} &= i_q - j i_d
\end{aligned}
$$

- Vector representation:

$$\bar{I} = \bar{I}_+ e^{+j\omega t} + \bar{I}_- e^{-j\omega t} = \bar{I}_p + \bar{I}_n$$

$$\bar{I}_+ = \tfrac{(1+\alpha)}{2} - j\tfrac{(1-\alpha)}{2\sqrt{3}}$$

$$\bar{I}_- = -\tfrac{(1-\alpha)}{2} - j\tfrac{(1-\alpha)}{2\sqrt{3}}$$

- Transform to $+\omega$ and $-\omega$ ref. frames:

$$
\begin{aligned}
\bar{I}^e_{+1} &= \bar{I} e^{-j\omega t} = \bar{I}_+ + \bar{I}_- e^{-j2\omega t} \\
\bar{I}^e_{-1} &= \bar{I} e^{+j\omega t} = \bar{I}_+ e^{+j2\omega t} + \bar{I}_-
\end{aligned}
$$

- Low pass filtering of $\bar{I}^e_{+1}$ yields $\bar{I}_+$.
- Low pass filtering of $\bar{I}^e_{-1}$ yields $\bar{I}_-$.
- Transform $\bar{I}_+$ from $+\omega$ frame to stationary frame yields the positive sequence vector $\bar{I}_p$.

$$\bar{I}_p = \bar{I}_+ e^{+j\omega t}$$

- Transform $\bar{I}_-$ from $-\omega$ frame to stationary frame yields the negative sequence vector $\bar{I}_n$.

$$\bar{I}_n = \bar{I}_- e^{-j\omega t}$$

- Transforming $\bar{I}_p$ to the *a-b-c* reference frame yields the positive sequence current components.
- Transforming $\bar{I}_n$ to the *a-b-c* reference frame yields the negative sequence current components.

EE 675: Microprocessor Applications in Power Electronics

Harmonic and Unbalanced Current Extraction
Sine- and Cosine-wave generation

Summary of Component Extraction Method

- Measure currents i_a, i_b and i_c.
- Transform them to the stationary *d-q* reference frame to get the stationary frame current vector.
- Transform the current vector to all appropriate rotating reference frames.
- Pass the transformed vectors through low pass filters.
- Transform the filtered quantities back to the stationary reference frame.
- Calculate the positive and negative sequence components.

Question: How to obtain the values of $\cos m\omega t$ *and* $\sin m\omega t$ *needed for transformations to the rotating reference frames?*

Answer: Implement a harmonic oscillator on a DSP.

Harmonic Oscillator Implementation

- Oscillator equations:

$$\dot{x} = +\omega y \qquad x(0) = 0$$
$$\dot{y} = -\omega x \qquad y(0) = 1$$

- This gives:

$$x = \sin \omega t$$
$$y = \cos \omega t$$

- Discrete time system:

$$x_{n+1} = x_n + \omega T \times y_n; x_0 = 0$$
$$y_{n+1} = y_n - \omega T \times x_{n+1}; y_0 = 1$$

- T is the time step with which these discrete time equations are solved.
- ω, the desired angular frequency, is the input to the oscillator.

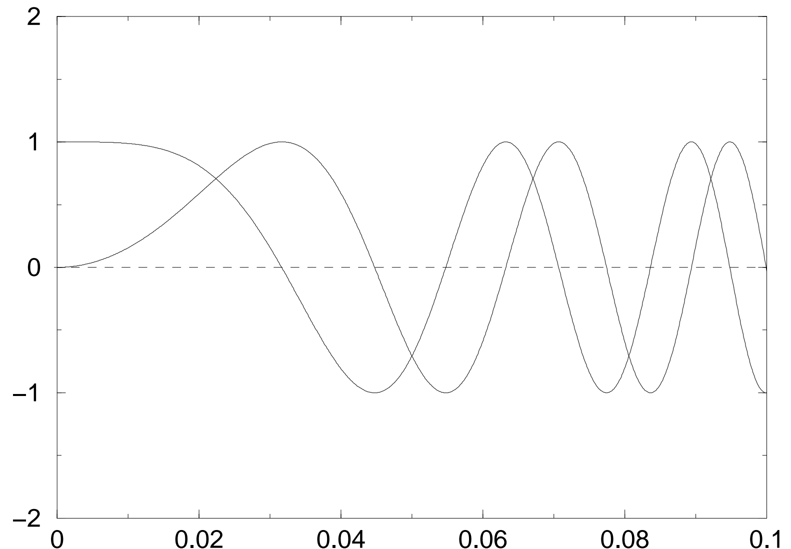
- TMS320VC33 assembly code:

```
ldf   @sin,r0
mpyf  @wdt,r0
negf  r0,r0
addf  @cos,r0
stf   r0,@cos
mpyf  @wdt,r0
addf  @sin,r0
stf   r0,@sin
```

- This code is executed every T seconds.
- The input to the code is ωT.
- The asymmetry in the discrete time equations is for reasons of stability.
- Higher (harmonic) frequency oscillations are obtained by multiplying ωT with the harmonic number m.

Sine and Cosine Wave Generation

- The frequency ω is ramped up at a constant rate.
- The time step $T = 20\mu s$.
- The result is a sine wave and a cosine wave of continuously increasing frequency.
- In DSP implementation, ωT can be the output of a PLL synchronised with the line voltage.
- Initialisation: Write 1.0 in the memory location for *cos*, and 0.0 in the memory location for *sin*.

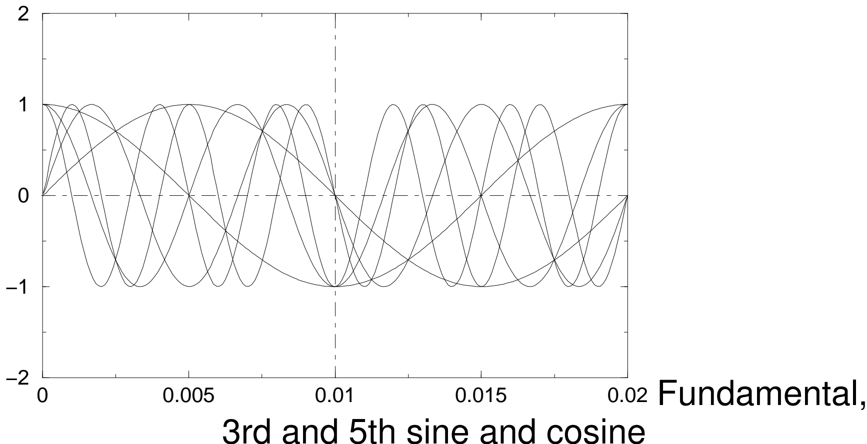


EE 675: Microprocessor Applications in Power Electronics

Harmonic Oscillator
Three-phase PLL

Oscillator Synchronisation

- Due to quantisation error, the oscillators for the harmonic frequencies tend to drift relative to the fundamental frequency oscillator.
- For higher order harmonics, the drift is higher.
- All oscillators are synchronised at the zero crossings of the fundamental sine wave.
- This involves re-initialising all sine values to zero, and all cosine values either to -1 or +1.

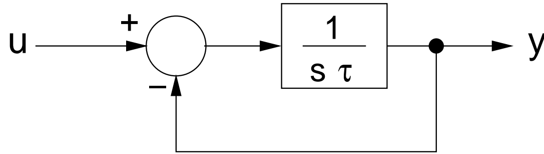


Fundamental, 3rd and 5th sine and cosine

- TMS320VC33 code:

```
 ldf   @sin_1,r0
 absf  r0
 cmpf  0.008,r0
 bp    _over_reset
 subf  r0,r0
 stf   r0,@sin_3
 stf   r0,@sin_5
 stf   r0,@sin_7
 stf   r0,@sin_9
 ldf   @cos_3,r0
 ldfp  1.0,r0
 ldfn  -1.0,r0
 stf   r0,@cos_3
 stf   r0,@cos_5
 stf   r0,@cos_7
 stf   r0,@cos_9
_over_reset:
```

First Order Low Pass Filter



- Transfer function:

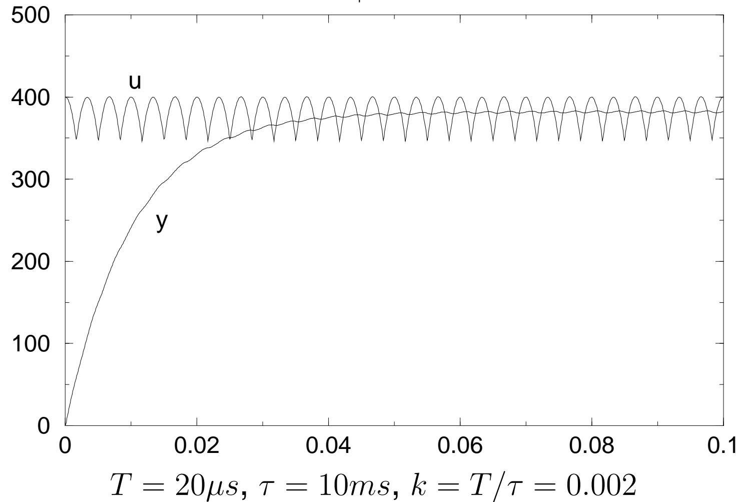
$$\frac{y(s)}{u(s)} = 1/(1 + s\tau)$$

- Discrete time implementation:

$$y_{n+1} = y_n + \frac{T}{\tau}(u_n - y_n)$$

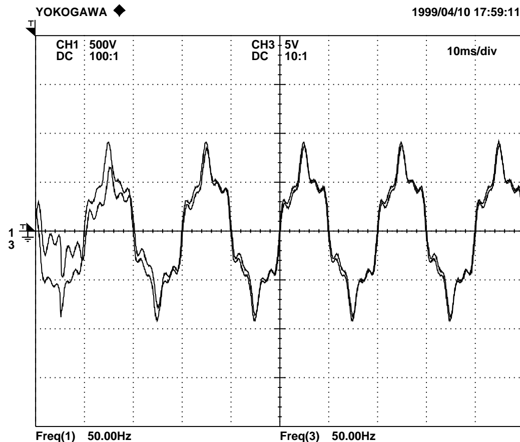
- TMS320VC33 code:

```
ldf   @u,r0
subf  @y,r0
mpyf  @k,r0
addf  @y,r0
stf   r0,@y
```

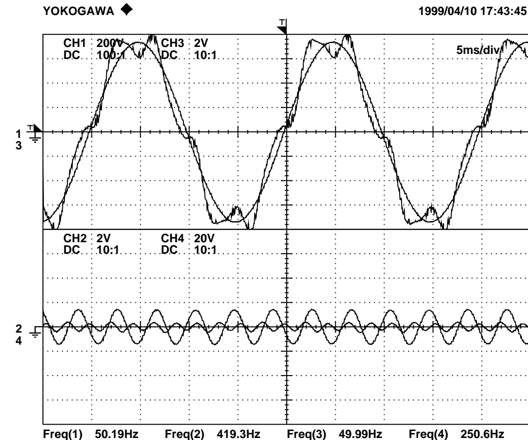


$T = 20\mu s$, $\tau = 10ms$, $k = T/\tau = 0.002$

Oscillograms from DSP Implementation



- Oscillogram shows input phase current i_a, and its reconstruction from its components. and also unbalanced harmonics.
- The positive and negative sequence sets for the fundamental and harmonics upto the 9th are extracted.

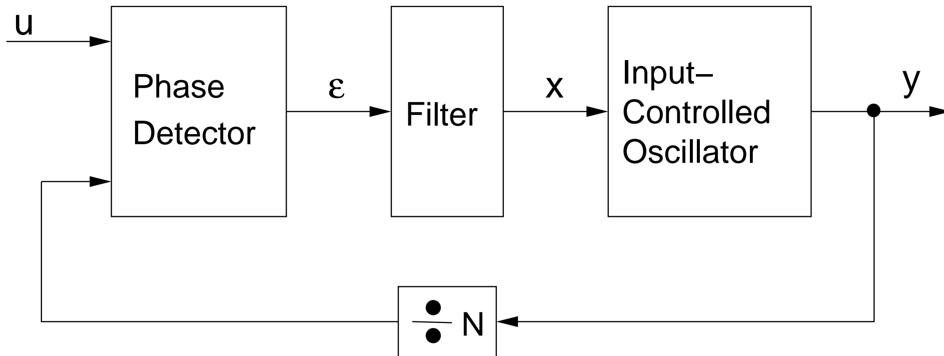


- Oscillogram shows input phase current i_c, and its extracted components.
- The fundamental component is superimposed on the actual current.
- The lower traces show the 5th and 9th harmonic components, reconstructed by adding their respective positive and negative sequence components.

EE 675: Microprocessor Applications in Power Electronics

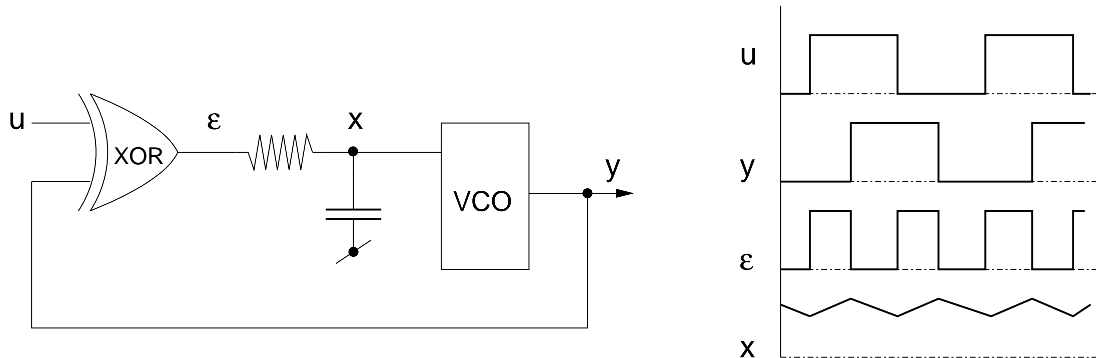
Three-phase PLL

Basic Phase Locked Loop



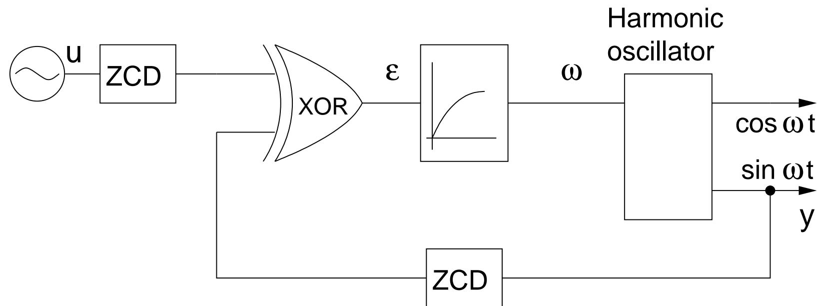
- Aim: To obtain the output *y* which has the same frequency as the input *u*, and has a constant phase relative to *u*.
- The signal *x* is obtained by filtering the phase error ϵ.
- The value of *x* determines the frequency of *y*.
- The signal *x* represents the frequency of *u*.
- Feature: if the frequency of *y* is divided by *N* before feeding to the phase detector, then this will be *N* times the frequency of *u*.

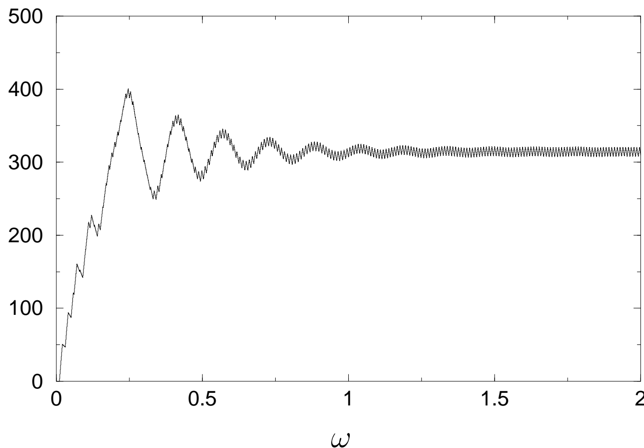
PLL Implementation with XOR Gate

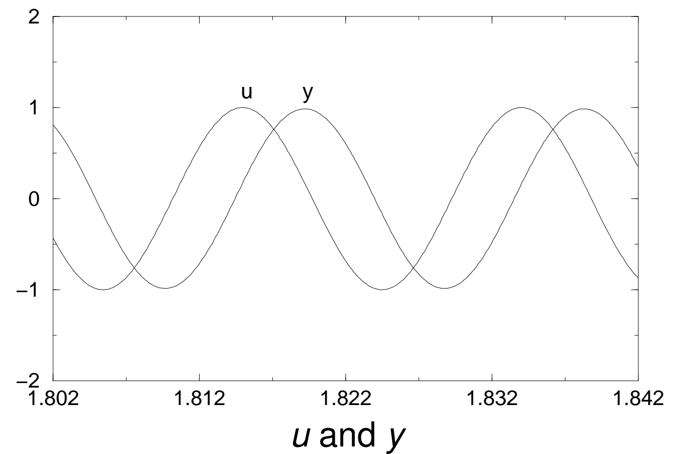


- The phase detector is an exclusive OR (XOR) gate.
- The filter is an RC circuit.
- The oscillator is a voltage controlled oscillator (VCO).
- The input *u* and output *y* have the same frequency.
- The output *y* lags *u* by $\pi/2$.
- We want sinusoidal inputs and outputs.

PLL Implementation with a Harmonic Oscillator







- All components of the PLL can be implemented on a DSP system.
- The output $\sin \omega t$ of the harmonic oscillator lags the input *u* by $\pi/2$.

Three Phase PLL

- Three phase PLL usage examples:
  - Frequency measurements.
  - Three phase static VAr compensators.
  - Voltage sag compensators.
  - Unbalance measurements.
  - Harmonic extraction.
  - Active filters.
- The PLL should use information from all three phases.
- It should operate in the presence of unbalances and harmonics.
- It should preferably not operate on the basis of zero crossings.
- It should have a specified dynamic response.
- We make use of reference frame transformations to implement a PLL.

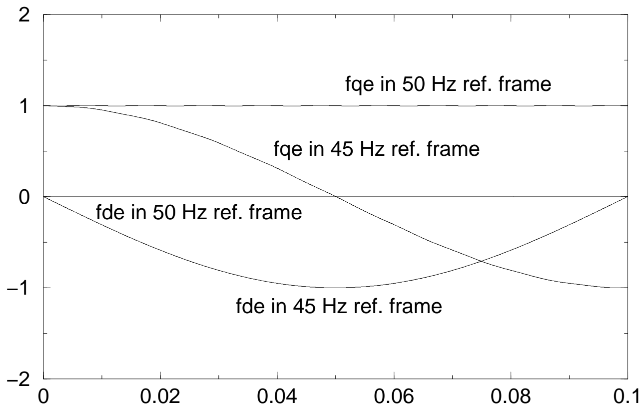
Example: PLL for Balanced Sinusoidal Voltages

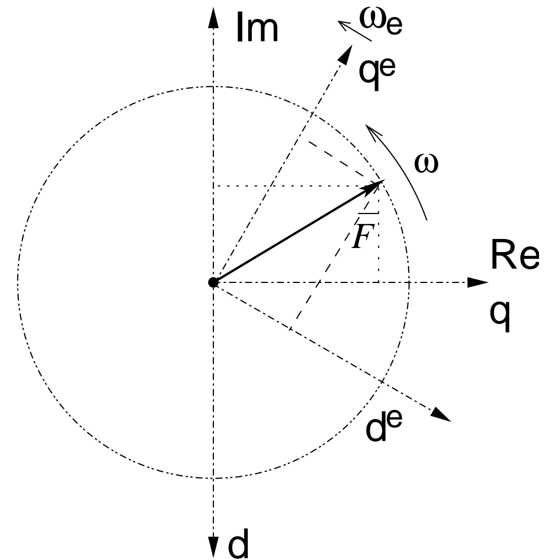
- Measured:

$$f_a = \cos\omega t$$
$$f_b = \cos(\omega t - 2\pi/3)$$
$$f_c = \cos(\omega t + 2\pi/3)$$

- Transformations:

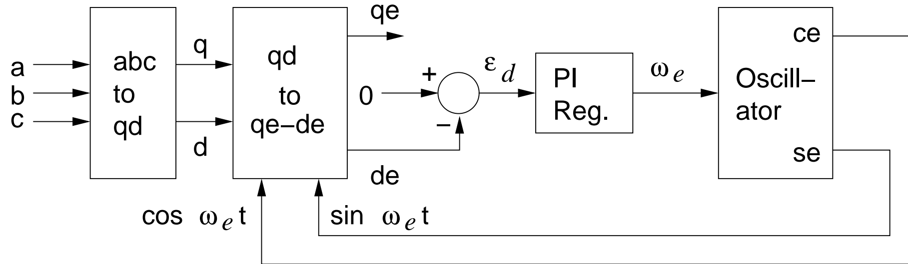
$$f_q = +\cos\omega t$$
$$f_d = -\sin\omega t$$
$$\bar{F} = \cos\omega t + j\sin\omega t = e^{j\omega t}$$
$$\bar{F}^e = e^{j(\omega-\omega_e)t}$$

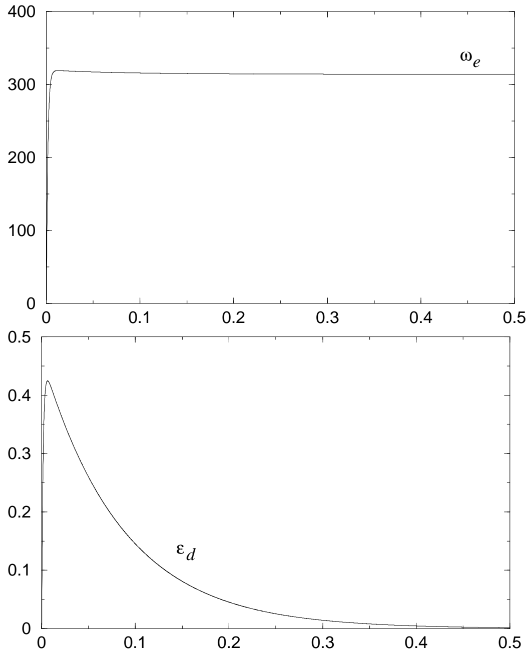




- Aim of the PLL: Find that rotating reference frame in which $f_d^e = 0$ and f_q^e is a constant.
- Find the angular frequency ω_e under closed loop control.

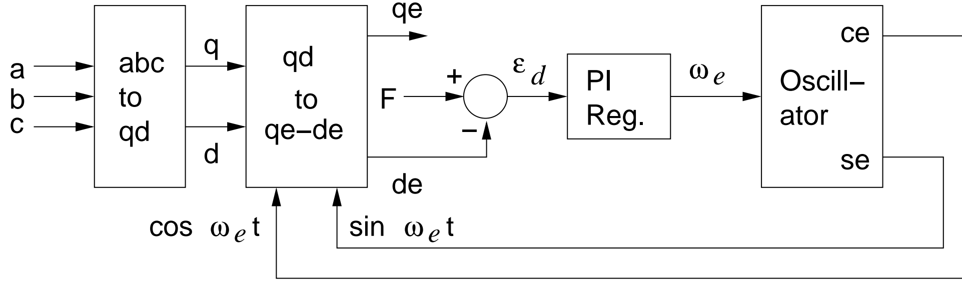
Three Phase PLL Implementation





- Also possible to make $q_e = 0$
- The PI controller sets $\omega_e T$ for the oscillator
- Can also be used for frequency measurement

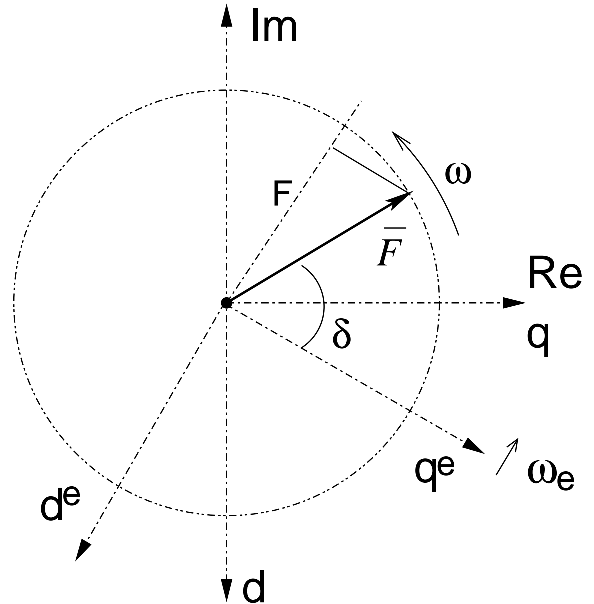
Three Phase PLL with Arbitrary Phase Angle



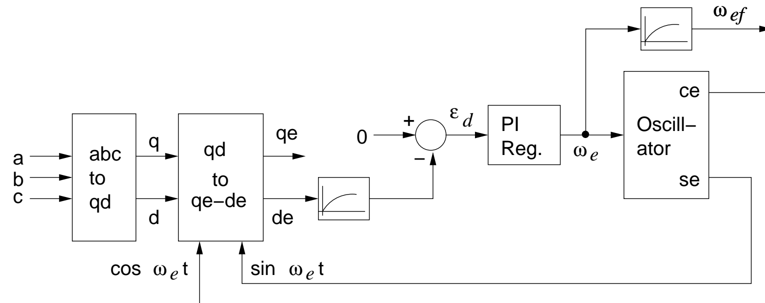
- Calculate set point F:

  F $= -(\sqrt{q^2 + d^2}) \times \sin\delta$

- This involves evaluating a square root and a sine function.
- This is computationally intensive. However, these functions can be evaluated on modern DSPs.
- It is desirable to make $\delta = 0$ if possible.



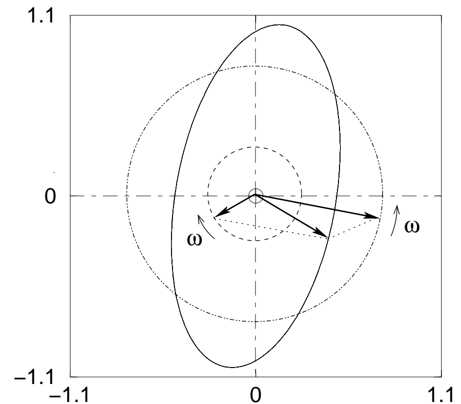
PLL Operation with Unbalanced Inputs



- Inputs:

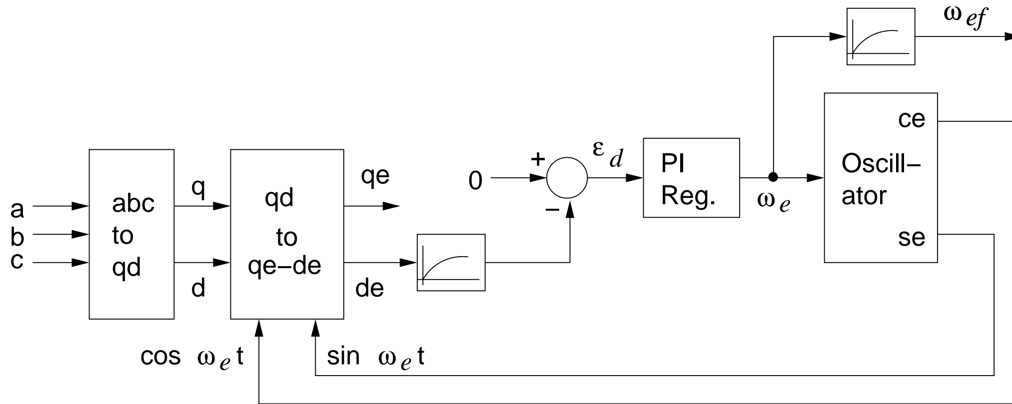
$$a = \alpha \cos(\omega t)$$
$$b = \cos(\omega t - 2\pi/3)$$
$$c = -(a + b)$$

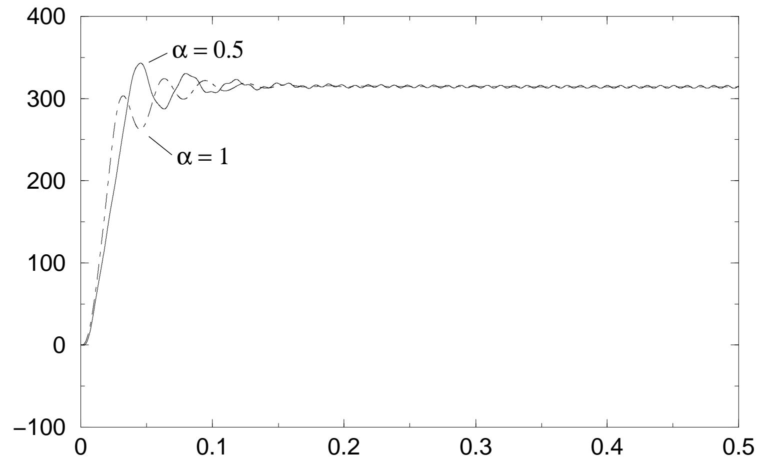
- Aim: To lock the PLL with the positive sequence component of the fundamental component of the input.
- A strong second harmonic component is present in f_d^e due to the fundamental negative sequence component.



The figure shows the vector loci for $\alpha = 0.5$. The vectors are shown at $t = 0$.

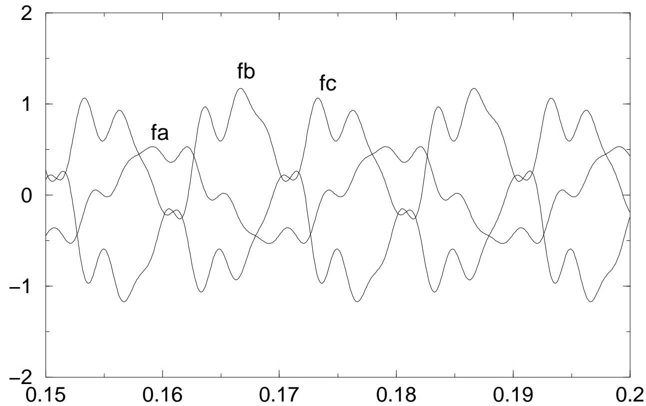
Simulation: PLL Operation with Unbalanced Inputs

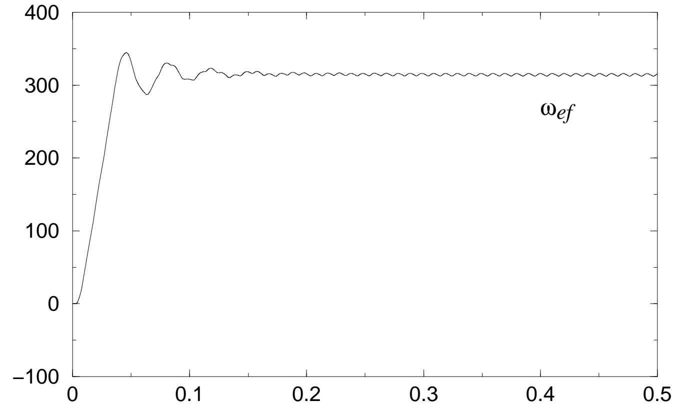


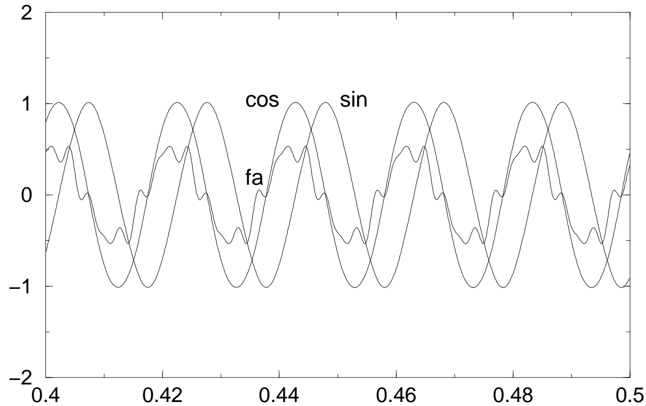


PLL frequency ω_{ef}. The input frequency is $\omega = 314.15\ rs^{-1}$.

Example: Unbalanced Fundamental and Harmonics







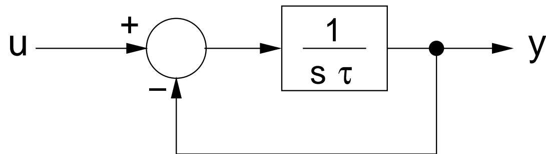
- The harmonics as well as the fundamental are unbalanced.
- The PLL locks to the positive sequence component of the fundamental component.
- The signals *sin* and *cos* can be used to effect rotation transformations that are synchronised with the fundamental positive sequence component.

Integration Methods for Real Time DSP Implementation

Integration Instances

- PI regulators.
- Discrete time implementations of continuous time filters.
- Integration of discretised dynamic plant models in sensorless control.
- Real time simulation on DSP platforms.
- *The basic question:*
  If the state vector of a system, and the vector of derivatives of the states, is known at the discrete time point $t(n)$*, what will the state of the system be at the discrete time point* $t(n+1) = t(n) + T$*?*
- In most power electronics applications, the interval T is constant.

Example: First Order Low Pass Filter

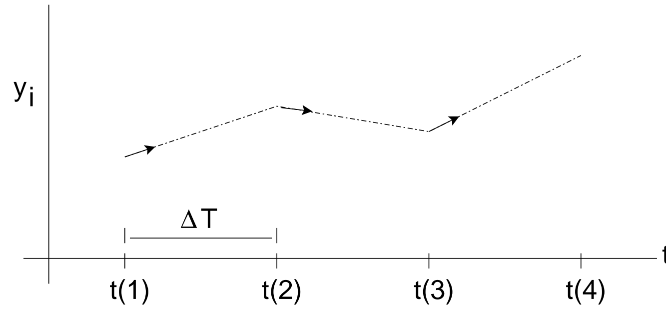


$\frac{y(s)}{u(s)} = 1/(1 + s\tau)$

- Known at time point $t(n)$:
  - Input $u(n)$.
  - Output $y(n)$.
  - Derivative $(u(n) - y(n))/\tau$.
- Calculate at time point $t(n + 1)$:
  - Output $y(n + 1)$.

One method to calculate the output is Euler's explicit integration method.

Euler's Explicit Method



- System of N first order ODEs:

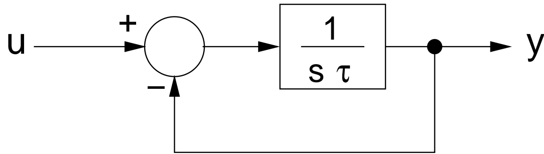
$$\dot{y}_i = f_i(t, y_1, \ldots, y_N), \quad i = 1 \ldots N$$
$$y_i(n+1) = y_i(n) + h f_i(t(n), y_1(n), \ldots, y_N(n))$$

- $h = T$ is the (constant) time step.
- The derivatives are calculated only at the start of the interval.

- Since all $y_i(n)$ are known at time point $t(n)$, the derivatives f_i can be calculated explicitly.
- The method has first order accuracy. The error is proportional to T.
- Not suitable for stiff systems unless T is much smaller than the fastest transient.

Low Pass Filter by Euler's Explicit Method

Unstable for $T > 2\tau$.

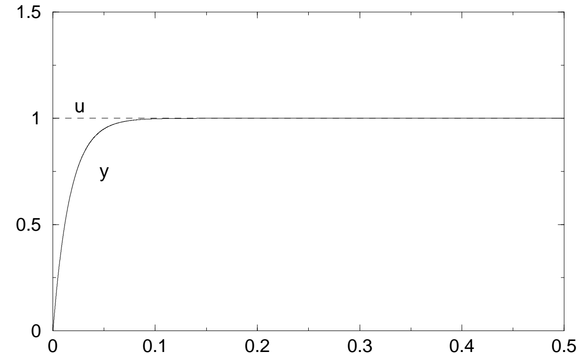


TMS320VC33 code:
(Executed every T s)

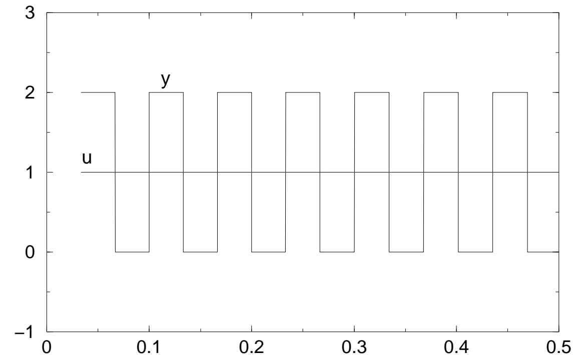
```
ldf   @u,r0
subf  @y,r0
mpyf  @k,r0
addf  @y,r0
stf   r0,@y
```

$$y(n+1) = y(n) + \frac{T}{\tau}\{u(n) - y(n)\}$$

$$y(n+1) = \left(1 - \frac{T}{\tau}\right) y(n) + \frac{T}{\tau} u(n)$$



$T = 20\mu s$, $\tau = 0.016666s$.



$T = 0.033333s$, $\tau = 0.016666s$.

Integration Methods for Real Time DSP Implementation – Continued

Euler's Implicit Method

- For the system of N first order ODEs:

$$\dot{y}_i = f_i(t, y_1, \ldots, y_N), \qquad i = 1 \ldots N$$

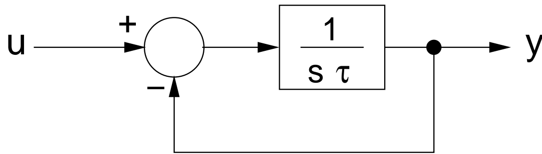
  evaluate the derivatives at the *new* location $y_i(n+1)$ instead of at $y_i(n)$.

- Also known as the Backward Euler method.
- Better suited for stiff systems than the Explicit (or Forward) Euler method.
- Stable even for large time steps.
- Solving the implicit equations to calculate the derivatives can be difficult. Example:

$$\begin{array}{l} \dot{y}_1 = f_1(y_1, y_2) \qquad ; \qquad \dot{y}_2 = f_2(y_1, y_2) \\ y_1(n+1) = y_1(n) + T \times f_1(y_1(n+1), y_2(n+1)) \\ y_2(n+1) = y_2(n) + T \times f_2(y_1(n+1), y_2(n+1)) \end{array}$$

- If the system of ODEs is nonlinear, then it needs to be linearised at every time point.

Low Pass Filter by Euler's Implicit Method

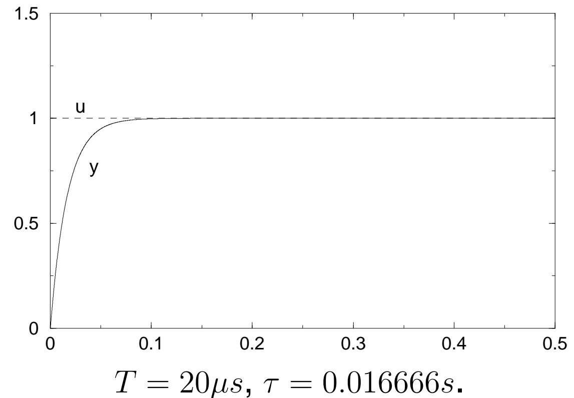


$$y(n+1) = y(n) + \frac{T}{\tau}\{u(n) - y(n+1)\}$$

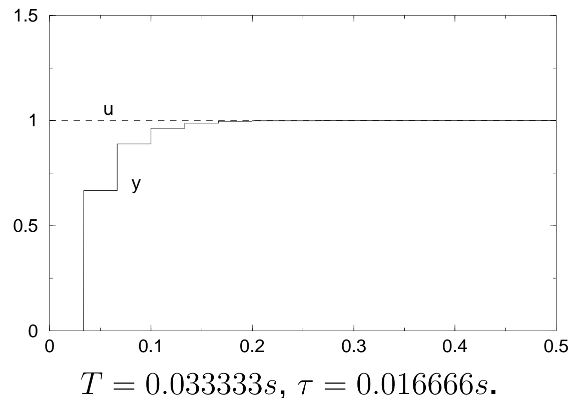
$$y(n+1) = \frac{1}{1+c}y(n) + \frac{c}{1+c}u(n)$$

$$c = \frac{T}{\tau}$$

- Stable for any time step.
- Easy to implement for a single ODE.
- Difficult to implement on a DSP in real time for a system of ODEs, as it involves the simultaneous solution of a system of algebraic equations.



$T = 20\mu s$, $\tau = 0.016666 s$.



$T = 0.033333 s$, $\tau = 0.016666 s$.

Heun's Method

- For the system of N first order ODEs:

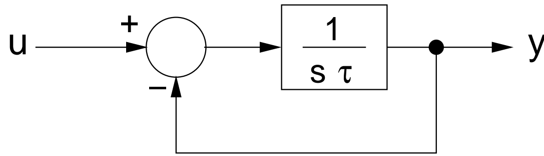
$$\dot{y}_i = f_i(t, y_1, \ldots, y_N), \qquad i = 1 \ldots N$$

evaluate the values of $y_i(n+1)$ in two steps.

$$\begin{aligned} y_i'(n+1) &= y_i(n) + T f_i(t(n), y_1(n), \ldots, y_N(n)) \\ y_i(n+1) &= y_i(n) + \tfrac{T}{2}\{f_i(t(n), y_1(n), \ldots, y_N(n)) \\ &\qquad\qquad + f_i(t(n+1), y_1'(n+1), \ldots, y_N'(n+1))\} \end{aligned}$$

- Also known as Modified Euler's Method.
- Has second order accuracy. The error is proportional to $(T)^2$.
- Computationally more intensive than Euler's Explicit Method.
- However, it permits the use of larger time steps than Euler's Explicit Method.

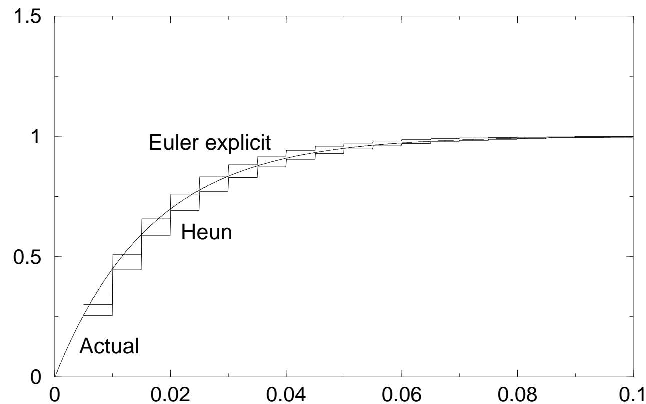
Low Pass Filter by Heun's Method



$$y'(n+1) = y(n) + c\{u(n) - y(n)\}$$
$$y(n+1) = y(n) + \frac{c}{2}[\{u(n) - y(n)\} + \{u(n) - y'(n+1)\}]$$
$$c = \frac{T}{\tau}$$

$$y(n+1) = (1 - c + \tfrac{c^2}{2})y(n) + (c - \tfrac{c^2}{2})u(n)$$

- Unstable for $T > 2\tau$
- Better accuracy and stability than Euler's Explicit Method.
- Computationally more intensive.



$T = 5ms,\ \tau = 0.016666s$

Harmonic Oscillator by Euler's Explicit Method

- Continuous time:

$$\dot{x} = +\omega y$$
$$\dot{y} = -\omega x$$

- Discrete time:

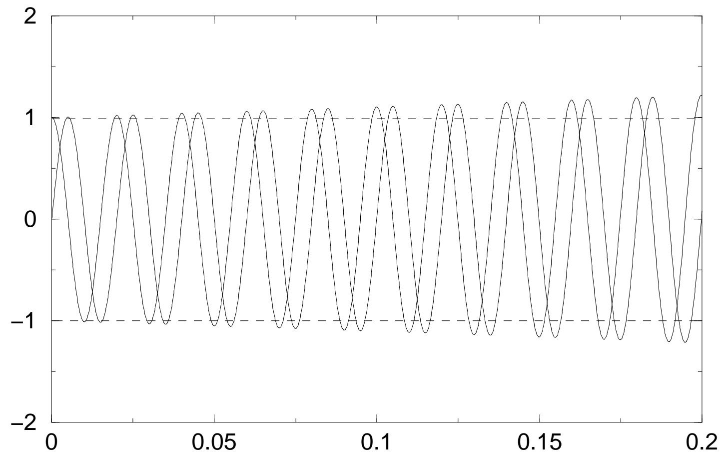
$$\begin{pmatrix} x(n+1) \\ y(n+1) \end{pmatrix} = \begin{pmatrix} 1 & h \\ -h & 1 \end{pmatrix} \begin{pmatrix} x(n) \\ y(n) \end{pmatrix}$$

$$x(0) = 0 \;\; ; \;\; y(0) = 1$$
$$h = \omega T$$

- Eigenvalues:

$$z = 1 \pm j\omega T$$

- System is unstable.



Harmonic Oscillator by Euler's Implicit Method

- Solve these simultaneous equations:

$$x(n+1) = x(n) + hy(n+1)$$
$$y(n+1) = y(n) - hx(n+1)$$

- Conditions:

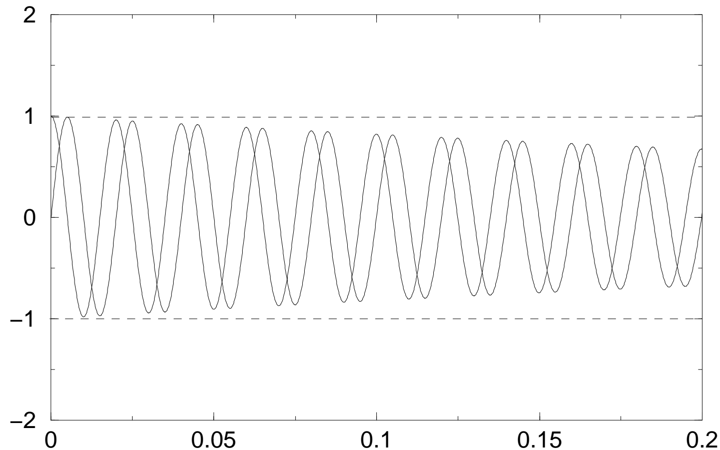
$$x(0) = 0 \ ; \ \ y(0) = 1$$
$$h = \omega T$$

$$\begin{pmatrix} x(n+1) \\ y(n+1) \end{pmatrix} = \begin{pmatrix} 1/(1+h^2) & h/(1+h^2) \\ -h/(1+h^2) & 1/(1+h^2) \end{pmatrix} \begin{pmatrix} x(n) \\ y(n) \end{pmatrix}$$

- Eigenvalues:

$$z = (1 \pm jh)/(1+h^2)$$

- System is stable.



Harmonic Oscillator: Practical Implementation

Discrete system:

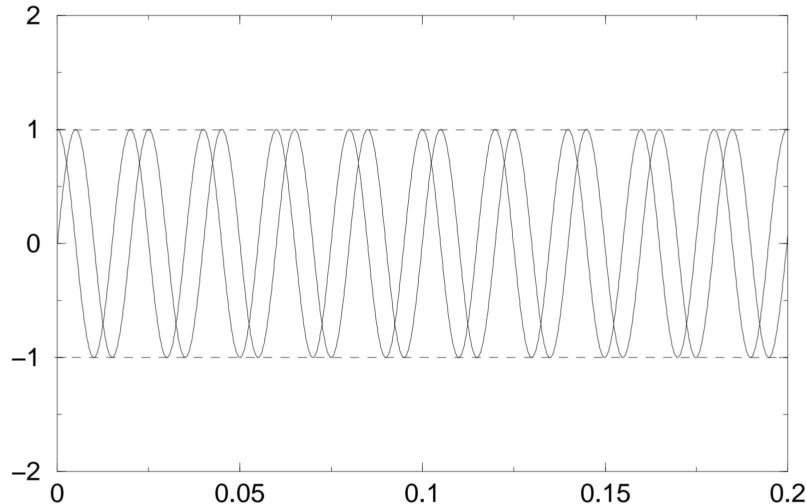
$$x(n+1) = x(n) + hy(n)$$
$$y(n+1) = y(n) - hx(n+1)$$

$$x(0) = 0; y(0) = 1$$
$$h = \omega T$$

$$\begin{pmatrix} x(n+1) \\ y(n+1) \end{pmatrix} = \begin{pmatrix} 1 & h \\ -h & 1-h^2 \end{pmatrix} \begin{pmatrix} x(n) \\ y(n) \end{pmatrix}$$

Eigenvalues:

$$z = \left(1 - \frac{h^2}{2}\right) \pm jh\sqrt{1 - \frac{h^2}{4}}$$



Trapezoidal Rule

- For the system of N first order ODEs:

$$\dot{y}_i = f_i(t, y_1, \ldots, y_N), \quad i = 1 \ldots N$$

  take the average of the derivatives at n and $n+1$.

- One of the *Newton-Cotes Formulas*.
- Stable even for large time steps.
- Solving the implicit equations to calculate the derivatives can be difficult. Example:

$$\begin{array}{l} \dot{y}_1 = f_1(y_1, y_2) \qquad ; \qquad \dot{y}_2 = f_2(y_1, y_2) \\ y_1(n+1) = y_1(n) + \frac{T}{2} \times [f_1(y_1(n), y_2(n)) + f_1(y_1(n+1), y_2(n+1))] \\ y_2(n+1) = y_1(n) + \frac{T}{2} \times [f_2(y_1(n), y_2(n)) + f_2(y_1(n+1), y_2(n+1))] \end{array}$$

- If the system of ODEs is nonlinear, then it needs to be linearised at every time point.
- Roger Cotes (1682 – 1756): English mathematician, close collaborator of Isaac Newton. Proofread Newton's *Principia Mathematica*.

Two-step Adams-Bashforth Method

$$\begin{aligned} y_i(n+1) &= y_i(n) + T\{\frac{3}{2} f_i(y_1(n), \ldots, y_N(n)) \\ &\quad -\frac{1}{2} f_i(y_i(n-1), \ldots, y_N(n-1))\} \end{aligned}$$

- The slope used is the weighted average of the slope at the previous time point, and the one before that
- Second order accuracy
- How to start the integration at $n = 0$?
- One way: Use Euler's method to calculate the value of $y_i(-1)$