Application note on REALIZER

Title: New features described
Date: 9 February 1999
Our ref. 0371700
Subject: Version 4.00
Author: René Balvers

Introduction

This document explains how Realizer® Version 4 generates code and how this code is structured.

It describes the various properties of the parameters and algorithms that Realizer uses to generate the code.

Realizer technology is based on the demands that have been identified in the world of embedded applications. Especially the need for a comfortable tool that would deal with the timing aspects of such applications is addressed with the availability of Realizer® Version 4.

they are programmed is mandatory. The target audience are readers who are interested in the working principles of Realizer® Version 4 or are users of earlier versions of Realizer.

The reader is supposed to have elementary experience in programming in assembly and has dealt with timing aspects of microcontroller applications.

All example schemes are for illustration purposes only. The resulting code shown is literally as generated by Realizer for the related schemes.

For reasons of convenience, all the code has been generated for STM's ST62 series of microcontrollers. Please note that Realizer can generate code for more than just this controller

architecture. For information about other supported microcontrollers, please visit our web site: www.actum.com.

Scope

This document assumes that the reader is either a user of Realizer or is interested in the working principles of Realizer. Basic knowledge about microcontrollers and the way

Standard Realizer generated code

Typically, the code generated by Realizer has a structure as shown in figure 1.

Realizer can apply optimizations while analysing the design and generating the code. The next sections of this document describe each of these optimisations.

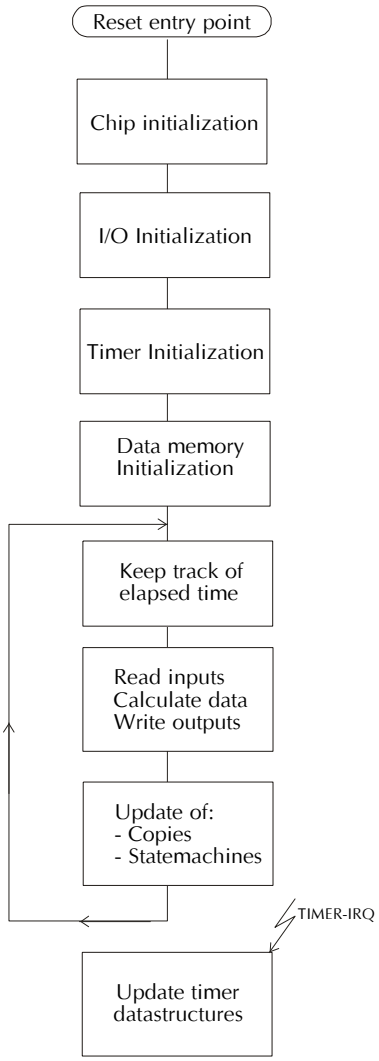

Figure 1 - Overall structure of Realizer generated code

New Extensions for Realizer® Version 4

With the release of V4 of Realizer a new and extended set of optimizations are available. Also known as Extensions, these features challenge even the most experienced assembly programmers.

The Version 4 Extensions can be listed as:

1. Periodic execution
2. Timed Interrupt execution
3. Subscheme input change
4. Compile-time operating system generation
5. Advanced hardware control

All the Version 4 Extensions are geared towards the same goal of the previous versions: To provide the users with a tool which realises most of the microcontroller applications in a very efficient and convenient way, without the need to be an expert on assembly language.

However the Version 4 Extensions operate on a much larger scale and with more impact. For its working Version 4 Extensions greatly co-operates closely with the designer/engineer.

Thus, Realizer gives your project with the best of both worlds: designers/engineers expertise and state of the art microcontroller design programming technology.

In the next sections the Version 4 Extensions by showing an example in which most of the Extensions are used.

Periodic execution

When the designer encounters a task that is only supportive but has to be serviced regularly, he now can put this information into the design. The following example shows this.

The application is a room temperature controller (thermostat). It should measure the room temperature, compare it with the required temperature, and take action if it is too low.

Figure 2 shows a part of this scheme. The complete design can be downloaded from our web-site: www.actum.com.

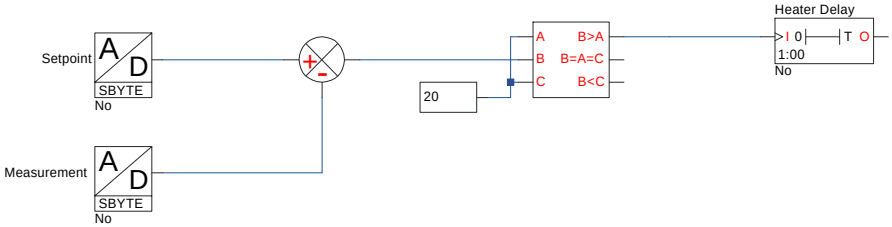

Figure 2 - Room temperature controller (part of ~)

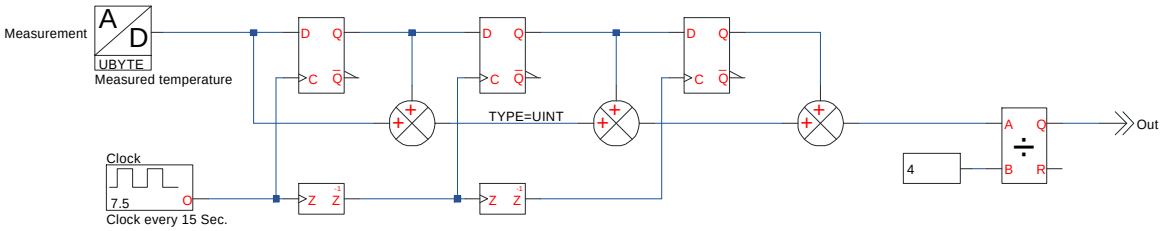

Figure 3 - Basic moving average filter

In this figure we see that after the compare using a subtraction, filtering is applied to prevent the controller from oscillating when differences are too small.

The filtering is built with the histeresis comparator and the Heater Delay timer of 1 second.

Please note that the constant of 20 at the input of the comparator is the equivalent of about 0.5 °C, which is determined by the electronic circuit that drives the input pin of the chip.

To increase the accuracy of such a controller one would like to have a moving average filter that would sample the temperature every 15 seconds and apply a moving average on those 4 samples.

A basic scheme (using Realizer V3.2 or earlier) that would do that, looks like the one shown in figure 3.

It shows that each program loop (please refer to figure 1 for an overview of the code generated by Realizer) in this scheme is recalculated. One could very easily get the impression that a lot of processor time is wasted **since it is "obvious" that recalculation is only needed once every 15 seconds**.

The Version 4 Extensions allow designers to put such information into the design.

In figure 4 we see that extra information is put into the scheme at the lower left corner of the drawing area. This is done by a right mouse click on the sheet itself. Just like an input symbol is connected to one of the physical pins of the microcontroller in the connection dialogue box, a *periodic event* is put onto the sheet.

The following figure shows this dialogue box (figure 5).

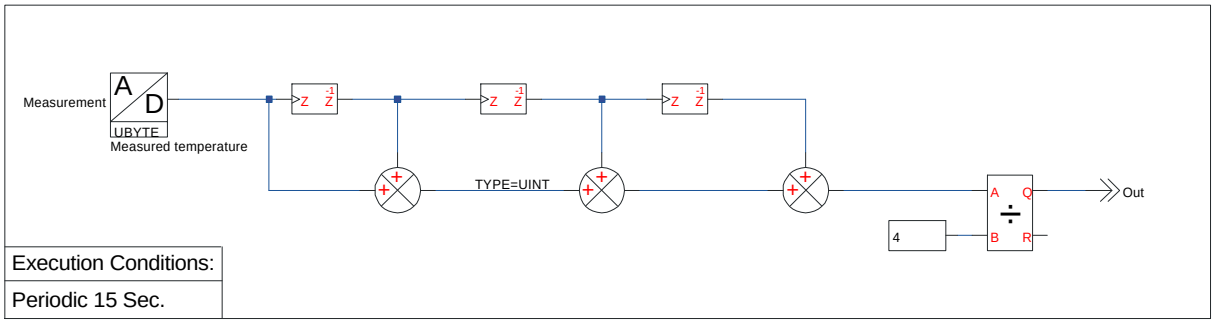

Figure 4 - Moving average filter in Version 4

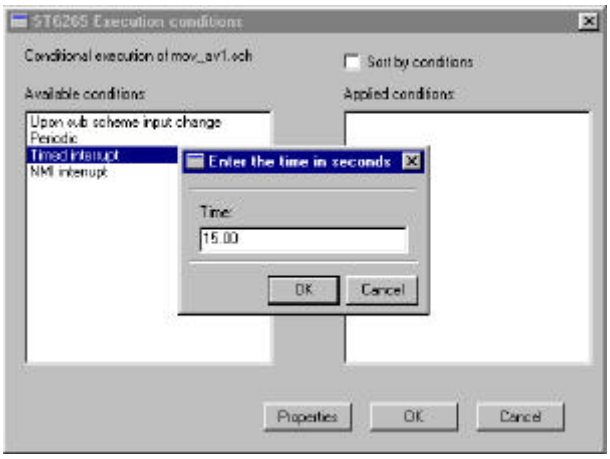

Figure 5 - Event Connection dialogue box (to the scheme)

The result is that optimised code is generated according to this information. The following figure shows the structure of the optimised code (figure 6).

Just as an experienced assembly programmer would do, it will keep track of time and, when time has elapsed, it will execute the appropriate code. All this is being generated automatically by Realizer® Version 4.

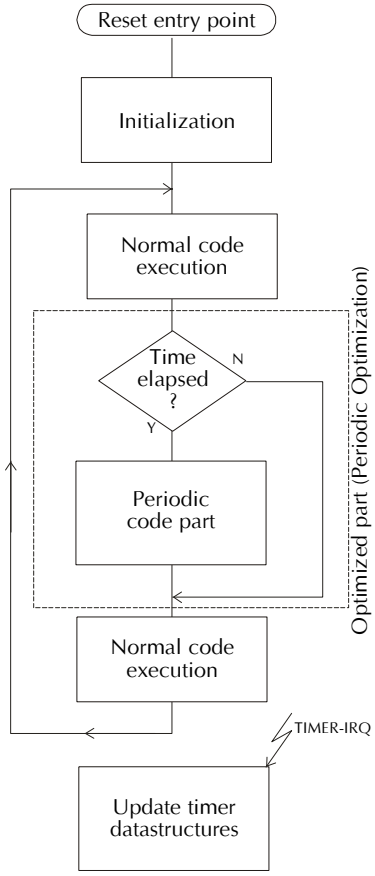

Figure 6 - Structure of Realizer code using the Periodic-event

Timed Interrupt execution

When writing assembly code you would use a timer interrupt to generate events on a regular basis to schedule tasks that need to be done at a specified interval.

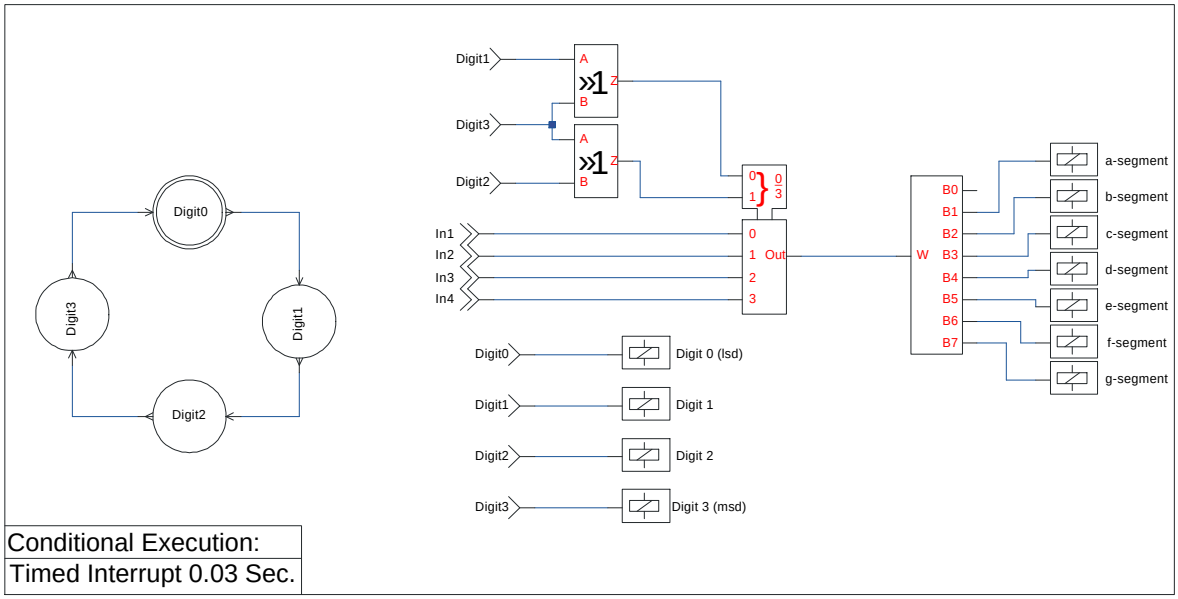

Figure 8 - Multiplexing LED display, drawn in Realizer® Version 4

A good example to show this is the multiplexing of an LED-display. Saving valuable I/O resources, a combination of 4 digit outputs and 7 segment outputs plus an appropriate amount of software would do that job. The human eye would not see the switching of the display refreshment as long as the software is fast enough.

The following scheme shows how the Realizer® Version 4 design looks like (figure 7).

On the scheme an event called Timed Interrupt is applied, notifying Realizer® Version 4 to generate code that would execute this subscheme every 30 milliseconds. This is shown in the drawing at the lower left-hand corner. The multiplexing rhythm is thus fixed at 30 mSec. The state machine (state diagram) will advance to the next state each time the scheme is executed, selecting a different pattern at the segments output and driving a different digit to the active state.

The code generated by Realizer® Version 4 has the structure as shown in figure 8. Comparing the structures of figures 1, 6 and 8, you can see the influence of the event attributes to the structure of the generated code.

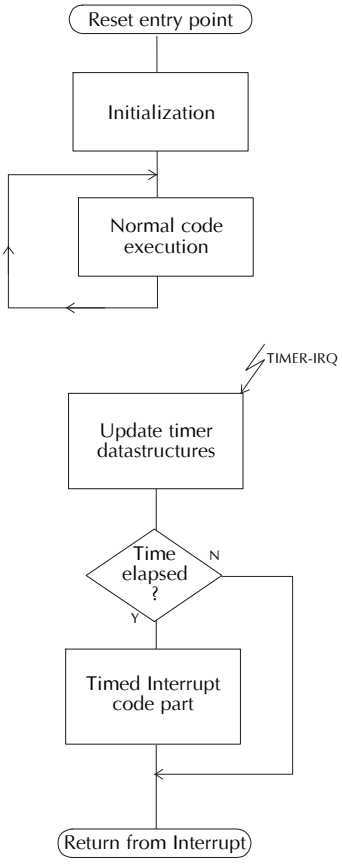

Figure 7 - Structure of Timed interrupt execution

Subscheme Input Change

Another optimization in Realizer® Version 4 is the execution of code upon data change.

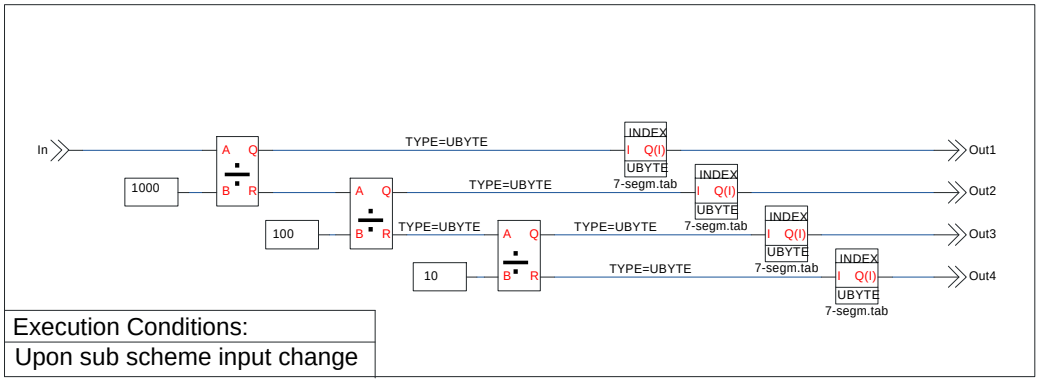

Figure 9 - Convert to 7-segment data only upon input change

Realizer® Version 4 handles this in the same transparent manner. By applying an attribute to the subscheme, the particular subscheme is only executed when the inputs of that subscheme have shown a change with respect to the previous execution.

So if you can identify parts of a design that depend on information that does not change a lot, you can use this optimization to increase the execution speed of the overall design.

The following example shows this:

To show the value of an engine speed indicator (4 digits showing the RPM of the engine), you need to extract the digit information from the measured value, see figure 9.

For this you need 4 dividers and 4 index tables. The tables convert the digit value (0..9) to the segment information for the displays.

The structure of the code is similar to the Timed event, but the decision is made upon the status changes on the input of the subscheme. See figure 10.

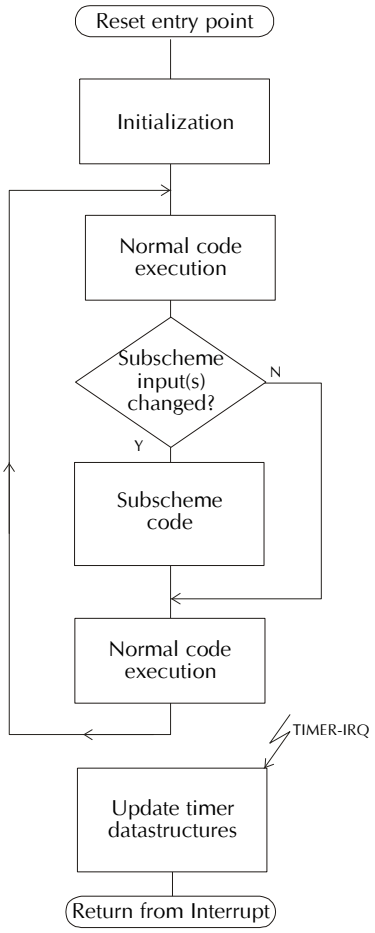

Figure 10 - Code structure for Subscheme Input Change

Compile time Operating System Generation

The advantages of the new Realizer V4 Extensions are that designer and computer software co-operate each in their own field of expertise. The designer simply puts in the data about the application, Realizer takes care of code generation and optimizations.

When dealing with (larger) computer systems, this attitude has resulted to the development and availability of operating systems and real-time kernels. Not only for desktops but for industrial controllers as well.

With the availability of smaller microcontrollers, developers of tools have tried to optimize the size and resource claims of their real time kernels. Selecting a kernel for specific applications is a process of making trade-offs over and over again.

Now with Realizer V4 it is has become a simple reality that the design tool incorporates the generation of a real-time kernel depending on the application.

It takes care of:

- Device drivers
- I/O register initialisation
- Interrupt serv. routines
- Data synchronisation (ISR <-> normal code)
- Task synchronisation
- Re-use of data (index tables, etc.)
- Allocation of processing power
- And much more..

This is all being determined at compile time.

This is what we at Actum Solutions call:

Application Specific Kernel® = ASK®

On the fly, at compile time, Realizer generates the optimal kernel for that particular application. So there is nothing too much and nothing too little in your kernel.

Examples

The pages in the back of this application note show some advanced schematic designs in which the features of Realizer® Version 4 are show. Most of these examples can be found on our web site: www.actum.com.

Pictures may tell you more than many paragraphs of text.

Conclusion

With the optimizations implemented in Realizer V3.20, a significant higher efficiency can be reached.

With the Realizer® Version 4 Extensions, Realizer is reaching the level of efficiency (code and data sizes) of the experienced assembly programmer. For the first time in computer technology history, there is a tool that programs your microcontroller as you would do it yourself!

Please note that Realizer can only excel in this task by using the information provided by the designer. With the availability of powerful desktop computers, the “cost” of compile cycles has become a trivial.

It is important to realise that the optimizations never interfere with the functionality of the design. So one can experiment freely to determine the right combination to trade off the various gains using the optimizations, either for code size, data memory size, or speed.

For more information please visit our web site at www.actum.com, send an e-mail to info@actum.com or fax us at: +31 (0)72 576 2559.

Appendix 1
Fast counter application - Schematics

The following description of an application is added to this application note to illustrate the features and properties of Realizer, especially the new features that come with V4.00.

The application is a pulse counting device used to measure the frequency of a digital signal. It uses the interrupt sensitive inputs of an ST6260 device from ST-Microelectronics.

It will filter the measured frequency so that a readable value is obtained. It converts these values into displayable data and then drives a 4 digit 7-segment LED-display. Figure 11 shows the basic block diagram.

The 4 major symbols are the sub-scheme symbols that represent the sub schemes.

The next diagrams show the 4 sub schemes in detail. Parts of these sub schemes are used in the first part of this application note. Please note that they are all made using Realizer® Version 4.

In appendix 2, the Realizer Report File (FASTCNT.RPF) is shown. It informs the designer what resources are used in which way. Also the code generated by Realizer is shown, with some comments added for your convenience.

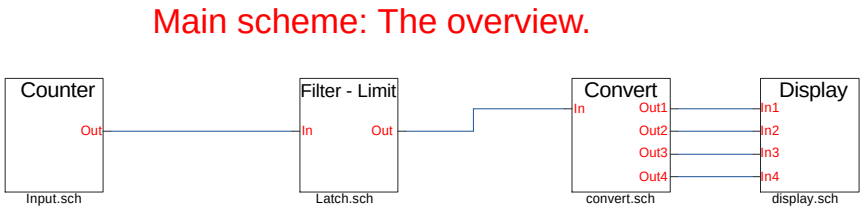

Figure 11 - Overview of the fast counter application

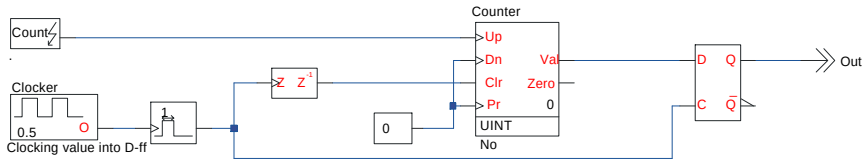

Figure 12 - Fast Counter: Input section

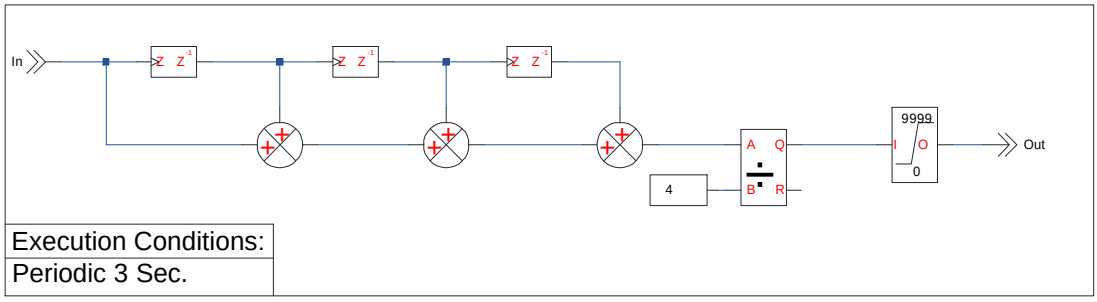

Figure 13 - Fast Counter: Filter and Limit section

Convert section: Creating the information for each digit.

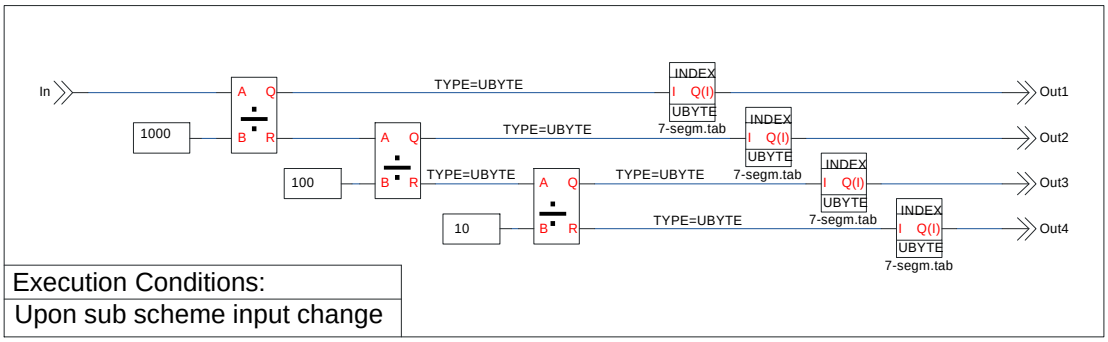

Figure 15 - Fast Counter: Conversion section

Display section: Driving the display in multiplexed mode.

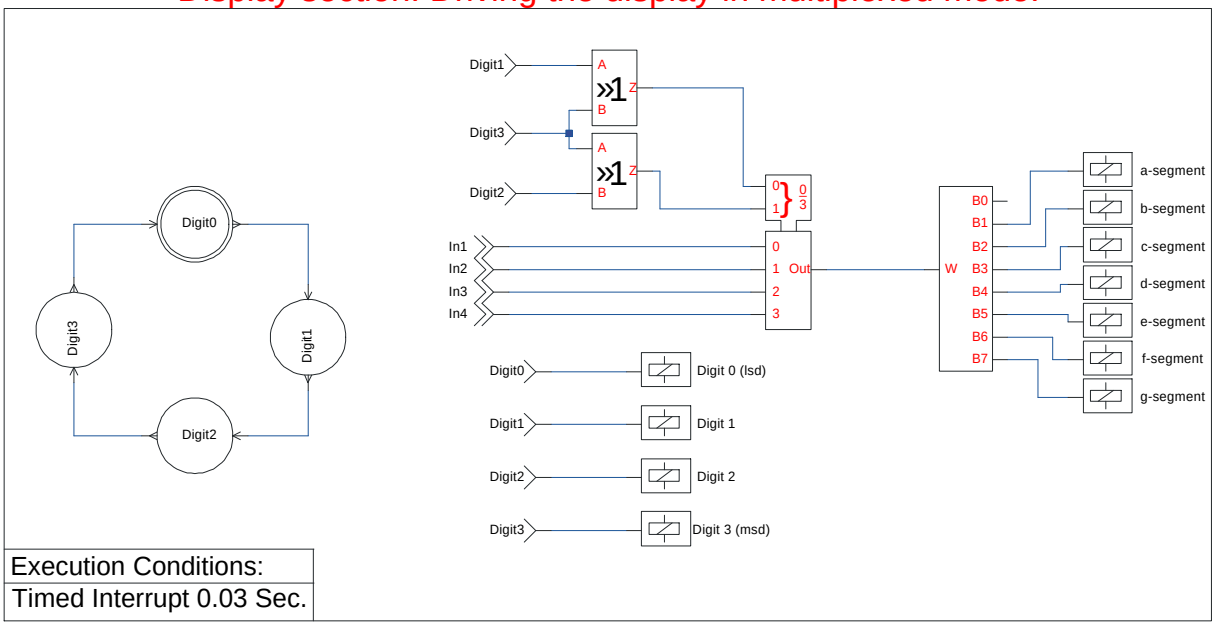

Figure 14 - Fast Counter: Display section

Appendix 2 - Fast counter application - Realizer Report File

This appendix shows the Realizer Report file that was generated by Realizer V4 for the sample application of appendix 1.

```
ST6260 Realizing Unit (V4.00) (c) 1990-99 Actum Solutions
Report file of project C:\REAL\EXAMPLES\V400\FastCnt\FastCnt.rpf
Scheme Version   : 1.03
Report timestamp : Fri Oct 23 15:38:55 1998
-------------------------------------------------------------------------

Schematic dependencies and events:
-------------------------------------------------------------------------

C:\REAL\EXAMPLES\V400\FastCnt\Fastcnt.sch
  Scheme: C:\REAL\EXAMPLES\V400\FastCnt\Latch.sch
    Event:  Periodic 3 Sec.
  Scheme: C:\REAL\EXAMPLES\V400\FastCnt\Input.sch
    Event:  Count=PB.2 interrupt
  Scheme: C:\REAL\EXAMPLES\V400\FastCnt\convert.sch
    Event:  Upon sub scheme input change
  Scheme: C:\REAL\EXAMPLES\V400\FastCnt\display.sch
    Event:  Timed Interrupt 0.03 Sec.


ST6260 (DIL20) connection overview:
-------------------------------------------------------------------------

Pin  Name       Alternative name      Type  I/O      Description
  1: PB.0       Digit 0 (lsd)       (BIT   Output), 20 mA sink open drain output
  2: PB.1       Digit 1             (BIT   Output), 20 mA sink open drain output
  3: TEST                           (            ), Test/program
  4: PB.2       Count               (BIT   Input ), interrupt
  5: PB.3                           (BIT   Input ), Not connected
  6: PB.6       Digit 2             (BIT   Output), 20 mA sink open drain output
  7: PB.7       Digit 3 (msd)       (BIT   Output), 20 mA sink open drain output
  8: PA.0       a-segment           (BIT   Output), 5 mA sink open drain output
  9: Vdd                            (            ), Power supply
 10: Vss                            (            ), Power supply
 11: PA.1       b-segment           (BIT   Output), 5 mA sink open drain output
 12: PA.2       c-segment           (BIT   Output), 5 mA sink open drain output
 13: PA.3       d-segment           (BIT   Output), 5 mA sink open drain output
 14: OSCin                          (            ), Oscillator
 15: OSCout                         (            ), Oscillator
 16: RESET                          (BIT   Input ), Active low
 17: NMI                            (BIT   Input ), Non Maskable Interrupt
 18: PC.4       f-segment           (BIT   Output), 5 mA sink open drain output
 19: PC.3       g-segment           (BIT   Output), 5 mA sink open drain output
 20: PC.2       e-segment           (BIT   Output), 5 mA sink open drain output

Hardware connections:
-------------------------------------------------------------------------

Symbolic name        H/W name   Type                          | Comment
Digit 0 (lsd)        PB.0       20 mA sink open drain output  |
Digit 1              PB.1       20 mA sink open drain output  |
Digit 2              PB.6       20 mA sink open drain output  |
Digit 3 (msd)        PB.7       20 mA sink open drain output  |
a-segment            PA.0       5 mA sink open drain output   |
b-segment            PA.1       5 mA sink open drain output   |
c-segment            PA.2       5 mA sink open drain output   |
d-segment            PA.3       5 mA sink open drain output   |
e-segment            PC.2       5 mA sink open drain output   |
f-segment            PC.4       5 mA sink open drain output   |
g-segment            PC.3       5 mA sink open drain output   |



Variable overview:
-------------------------------------------------------------------------

Total used bits              : 23
Total used events            : 5
Total used unsigned bytes    : 20
Total used signed bytes      : 0
Total used unsigned integers : 19
```

```
Total used signed integers   : 0
Total used longs             : 0


Memory overview:
------------------------------------------------------------------------

Total used RAM               : 73 byte (000H->03FH,085H->08CH)
Total used ROM               : 1714 byte (0080H->0732H) of 0F9DH

Note: The pins that are not used are defined as digital input with pull-up.
      The timer pin is configured as an output.
```

Appendix 3 - Fast counter application - Generated code

This appendix shows the code that Realizer V4 has generated for the sample application discussed in appendix 1.

```
; ST Realizer (V4.00) : generated ST6260 Code                    Standard heading for each generated
; File           : C:\REAL\EXAMPLES\V400\FastCnt\Fastcnt.asm     project
; Scheme Version : 1.03
; Date           : Fri Oct 23 15:38:50 1998
; Used variables : 52
; Used functions : 100


.VERS "ST6260"
.ROMSIZE 4
.DP_ON

.INPUT "Fastcnt.inc"                                            Include variables and definitions
.INPUT "C:\real\LIB\v400\ST6260.inc"                            Include symbol libraries
.INPUT "C:\real\LIB\v400\st62lib.mac"
.INPUT "C:\real\LIB\v400\st62xx.mac"
.INPUT "C:\real\LIB\v400\st62eepr.mac"


.ORG 00080H                                                     Start at first ROM location
RROMST:
                                                                Project data included in the code
.ASCIZ "Fastcnt"
.ASCIZ "1.03"

Reset:
  LDI IOR,IOR_MASK                                              Initialize chip
  LDI HWDR,0FFH

  LDI DRBR,010H
  LDI EECTL,000H

; DBG:  initialize PORTB events                                 Initialize I/O resources

; DBG:    Enable interrupt on PORTB.2
PortInit:
  LDI DDRA,00FH
;  LDI ORA,000H                                                 Optimized line
  LDI DRA,00FH
  LDI BUDRA,00FH
  LDI DDRB,0C3H
  LDI ORB,004H
  LDI DRB,0C3H
  LDI BUDRB,0C3H
  LDI DDRC,01CH
;  LDI ORC,000H                                                 Optimized line
  LDI DRC,01CH
  LDI BUDRC,01CH

Rtcinit:                                                        Initialize hardware timer
  LDI PSC,015H
  LDI TCR,0CFH
  LDI TSCR,06DH
  CLR TICK
  LDI IOR,(IOR_MASK | 010H)
  RETI

RamInit:                                                        Initialize data memory with 0
  LDI A,0
  LDI X,000H
  LDI Y,62
RamInit1:
  LD (X),A
  INC X
  DEC Y
  JRNZ RamInit1

  iconstw v1n27,2,4                                             Initialize special symbols
  iconstb v2n2,5,1,0                                            (presets, etc.
  idltchwbww v2n6,4,v2n7,2,1,y2n5,4,0,0
  iconstw v3n7,2,10
  iconstw v3n6,2,100
  iconstw v3n4,4,1000
```

```
  JP EIS03011
IS03011:
.BYTE  000H                                          This is the index table for 7
.BYTE  07EH                                          segment display
.BYTE  001H                                          Note that one table is used to
.BYTE  030H                                          implement the 4 index table symbols
.BYTE  002H
.BYTE  06DH
.BYTE  003H
.BYTE  079H
.BYTE  004H
.BYTE  033H
.BYTE  005H
.BYTE  05BH
.BYTE  006H
.BYTE  01FH
EIS03011:

RealMain:                                            This is where the real thing starts
Rtc:                                                 Keep track of time elapsed since
  LDI IOR,IOR_MASK                                   previous loop started
  LD A,TICK
  CLR TICK
  LDI IOR,(IOR_MASK | 010H)
  LD RTICK,A
RTIMEND:

  LD A,RTICK                                         Pass it to the application
  JRNZ IRTC0001
  JP RTCSKIP
IRTC0001:
; DBG:  Decrement 16 bit timers
  JRS 7,037H,RTC0001                                 Update timers if tick(s) was(were)
  sub2www 037H,4,RTICK,2,037H,4                      detected
RTC0001:

RTCSKIP:

RINPEND:

;    moved to TIMER1 interrupt:   CALL SUB0002
  LDI IOR,000H                                       Synchronize data between interrupt
  copyww y2n5,4,v0n5,4                               service routine and normal
  LDI IOR,010H                                       executed code

;    moved to PORTB interrupt:   CALL SUB0002

;    check for time-out                              TIMED part:
  JRR 7,AT00001,_UNI0000                             If 3 sec. have elapsed:
  LDI AT00001+0,1                                    Reload software timer
  LDI AT00001+1,43                                   and
  CALL SUB0001                                       execute the subscheme
_UNI0000:

;   Input change detection on v0n7                   SUBSCHEME INPUT CHANGE part:
  LD A,v0n7+0                                        is current value (msbyte)
  CP A,pv0n7+0                                       equal to previous one?
  JRNZ EXEC0003                                      no: execute subscheme
  LD A,v0n7+1                                        is current value (lsbyte)
  CP A,pv0n7+1                                       equal to previous one?
  JRNZ EXEC0003                                      no: execute subscheme
  JP NOEX0003                                        yes: skip this subscheme
EXEC0003:
  CALL SUB0003                                       Call it
NOEX0003:

;   Disable interrupts for parameter passing         Pass parameters to the interrupt
  LDI IOR,IOR_MASK                                   driven display section
  copyww v0n9,2,x4n3,2                               ( the 4 digit data bytes )
  copyww v0n11,2,x4n5,2
  copyww v0n10,2,x4n4,2
  copyww v0n8,2,x4n2,2

;   Enable interrupts
  LDI IOR,(IOR_MASK | 010H)

;   moved to TIMER1 interrupt:   CALL SUB0004
ROUTPEND:
  copyww v0n7,4,pv0n7,4                              Update some previous values and
  LDI IOR,IOR_MASK                                   backed up data registers
  LD A,BUDRA
  LD DRA,A
  LD A,BUDRB
```

```
  LD DRB,A
  LD A,BUDRC
  LD DRC,A
  LDI IOR,(IOR_MASK | 010H)
  LDI HWDR,0FFH
  JP RealMain

SUB0001:                                                    Subroutine: Filter and Limit
  loopdelwww v0n5,4,sv1n22,4,v1n20,4
  loopdelwww v1n20,4,pv1n20,4,v1n21,4
  loopdelwww v1n21,4,pv1n21,4,v1n25,4
  add2www v0n5,4,v1n20,4,v1n23,4
  add2www v1n23,4,v1n21,4,v1n24,4
  add2www v1n24,4,v1n25,4,v1n26,4
  divwwww v1n26,4,v1n27,2,v1n29,4,0,0
  limfww v1n29,4,v0n7,4,0,9999
  copyww v0n5,4,sv1n22,4
  copyww v1n20,4,pv1n20,4
  copyww v1n21,4,pv1n21,4
  ret
                                                            Subroutine: Input, called from edge
SUB0002:   ; interrupt driven sub routine                   sensitive interrupt service routine
  oscfb v2n0,0,1,49,100,T02006,2                            and from timer interrupt serv. rout.
  edgebbb v2n0,0,1,pv2n0,1,1,v2n7,2,1
  loopdelbbb v2n7,2,1,pv2n7,3,1,v2n1,4,1
  eventb v2n8,6,1
  countfbbbbbbbwb
v2n8,6,1,pv2n8,7,1,v2n2,5,1,pv2n2,0,1,v2n1,4,1,v2n2,5,1,pv2n2,
0,1,v2n6,4,0,0,0,0,CT02000,4
  dltchwbww v2n6,4,v2n7,2,1,y2n5,4,0,0
  copybb v2n0,0,1,pv2n0,1,1
  copybb v2n7,2,1,pv2n7,3,1
  copybb v2n2,5,1,pv2n2,0,1
  ret

SUB0003:                                                    Subroutine: Convert
  divwwww v0n7,4,v3n4,4,v3n3,2,v3n1,4
  divwwww v3n1,4,v3n6,2,v3n17,2,v3n2,2
  divwwww v3n2,2,v3n7,2,v3n15,2,v3n8,2
  indtabww v3n8,2,v0n11,2,S03011,14,0                       ! only one table is used for the 4
  indtabww v3n15,2,v0n10,2,S03011,14,0                      symbols
  indtabww v3n17,2,v0n9,2,S03011,14,0
  indtabww v3n3,2,v0n8,2,S03011,14,0
  ret

SUB0004:                                                    Subroutine: Display
  stateoutb v4n22,1,1,st0,2,2
  stateoutb v4n21,2,1,st0,2,3
  or2bbb v4n21,2,1,v4n22,1,1,v4n1,3,1
  stateoutb v4n19,4,1,st0,2,1
  or2bbb v4n19,4,1,v4n21,2,1,v4n0,5,1
  mux2bbwwwww
v4n0,5,1,v4n1,3,1,x4n2,2,x4n3,2,x4n4,2,x4n5,2,v4n6,2
  bunpackwbbbbbbbb
v4n6,2,0,0,0,v4n43,6,1,v4n47,7,1,v4n42,0,1,v4n41,1,1,v4n44,2,1
,v4n45,3,1,v4n46,4,1
  digoutb v4n46,4,1,BUDRC,3,1
  digoutb v4n45,3,1,BUDRC,4,1
  digoutb v4n44,2,1,BUDRC,2,1
  digoutb v4n42,0,1,BUDRA,2,1
  digoutb v4n47,7,1,BUDRA,1,1
  digoutb v4n43,6,1,BUDRA,0,1
  digoutb v4n41,1,1,BUDRA,3,1
  stateoutb v4n12,5,1,st0,2,0
  digoutb v4n12,5,1,BUDRB,0,1
  stateoutb v4n13,6,1,st0,2,1
  digoutb v4n13,6,1,BUDRB,1,1
  stateoutb v4n14,7,1,st0,2,2
  digoutb v4n14,7,1,BUDRB,6,1
  stateoutb v4n15,0,1,st0,2,3
  digoutb v4n15,0,1,BUDRB,7,1
  stateminit st0,2,0
  stateminit st0,2,1
  stateminit st0,2,2
  stateminit st0,2,3
  stateminit st0,2,-1
  stateinit st0,2,0
  state st0,2,0,1
  stateend st0,2,0
  stateinit st0,2,1
  state st0,2,1,2
  stateend st0,2,1
  stateinit st0,2,2
```

```
  state st0,2,2,3
  stateend st0,2,2
  stateinit st0,2,3
  state st0,2,3,0
  stateend st0,2,3
  statemend st0,2,4
  LD A,BUDRA
  LD DRA,A
  LD A,BUDRB
  LD DRB,A
  LD A,BUDRC
  LD DRC,A
  ret

.IFC NDF Rtcint
Rtcint:                                                 TIMER interrupt service routine
  LDI TSCR,000H
  LDI PSC,015H
  LDI TCR,0CFH
  LDI TSCR,06DH
  INC TICK                                              Signal elapsed time to normal s/w
                                                        timers
; DBG:  Create normal stack push
  LD STACKA,A
  LD A,X
  LD STACKX,A
  LD A,Y
  LD STACKY,A
  LD A,V
  LD STACKV,A
  LD A,W
  LD STACKW,A
  LD A,REG0
  LD STACK0,A

; DBG:  Decrement interrupt timers                      Oscillator in input scheme
  JRS 7,T02006,TM1_0002
  decw T02006,2                                         Timed out?
  JRR 7,T02006,TM1_0002
  CALL SUB0002                                          Yes: run subscheme
TM1_0002:

; DBG:  Decrement prioritized timers                    Display subscheme
  decw AT00004,2
  JRR 7,AT00004,TM1_0003                                Time to be executed again?
  CALL SUB0004                                          Yes
  copyww 2,TCONST,AT00004,2
TM1_0003:

; DBG:  Create normal stack pop
  LD A,STACK0
  LD REG0,A
  LD A,STACKW
  LD W,A
  LD A,STACKV
  LD V,A
  LD A,STACKY
  LD Y,A
  LD A,STACKX
  LD X,A
  LD A,STACKA
  RETI
.ENDC


; DBG:  PORTA,PORTB interrupt service routine
.IFC NDF PORTAint
PORTAint:

; DBG:  Create normal stack push
  LD STACKA,A
  LD A,X
  LD STACKX,A
  LD A,Y
  LD STACKY,A
  LD A,V
  LD STACKV,A
  LD A,W
  LD STACKW,A
  LD A,REG0
  LD STACK0,A
  SET 6,v2n8                                            Set event to ON
  CALL SUB0002                                          execute subscheme
```

```
  RES 6,v2n8                                                    Reset event again

; DBG:  Create normal stack pop
  LD A,STACK0
  LD REG0,A
  LD A,STACKW
  LD W,A
  LD A,STACKV
  LD V,A
  LD A,STACKY
  LD Y,A
  LD A,STACKX
  LD X,A
  LD A,STACKA
  RETI
.ENDC

RROMEND:

.ORG 0FFEH                                                      Reset and interrupt vectors
  JP Reset

.ORG 0FF0H
  JP Rtcint

.ORG 0FF6H
  JP PORTAint
```