Atmel AVR1618: ATxmegaB ASCII Character Mapping

Features

- **ASCII character mapping available on Atmel® AVR® XMEGA® B microcontrollers**
- **Atmel XMEGA-B1 Xplained evaluation kit compatible**
- **On-board LCD display glass**
- **Structures within glass**
  - **7-segment numeric**
  - **14-segment alphanumeric**

1 Introduction

The ASCII character mapping built-in the LCD controller of Atmel AVR XMEGA B devices allows setting and clearing pixels of a 7-, 14-, or 16-segment character simply by writing the corresponding ASCII code.

This application note presents an example that demonstrates 7- and 14-segment character handling using the XMEGA-B1 Xplained kit.

Figure 1-1. XMEGA-B1 Xplained kit.

8-bit Atmel Microcontrollers

Application Note

Rev. 8451A-AVR-11/11

2 Environment

The demonstration is based on the Atmel XMEGA-B1 Xplained kit with Atmel ATxmega128B1 (www.atmel.com/xplained) and its dedicated LCD glass "*C42048A"*.

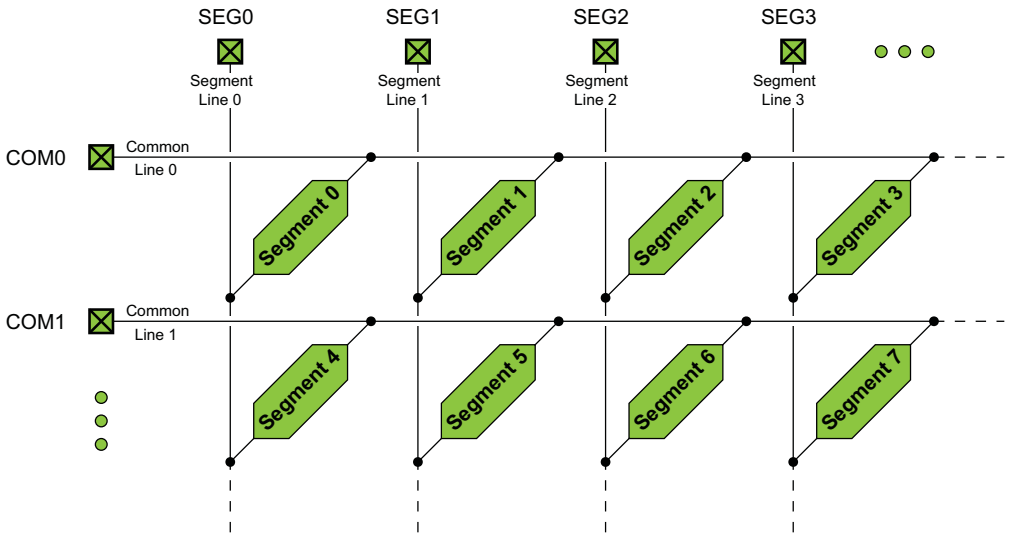
Firmware is available on Atmel AVR Software Framework and Atmel AVR Studio® 5 as the following examples:

- "LCD C42048A glass example – XMEGA-B1 Xplained – Atxmega128B1"
- "LCD controller example – XMEGA-B1 Xplained – Atxmega128B1"

Table 2-1. LCD definitions and acronyms.

LCD	Liquid crystal display. A passive display panel with terminal lines leading directly to a segment.
Segment or pixel	LCD panel active area which can be turned "ON or "OFF". Each segment has two lines. One is connected to a SEG line the other is connected to a COM line. All segments are connected in a matrix (see Figure 2-1).
COM	Common line or backplane line
SEG	Segment line
Duty	1/(number of COM lines on a LCD display)
Bias	1/(number of voltage levels used driving a LCD display -1)
DP	Decimal point

Figure 2-1. LCD matrix.

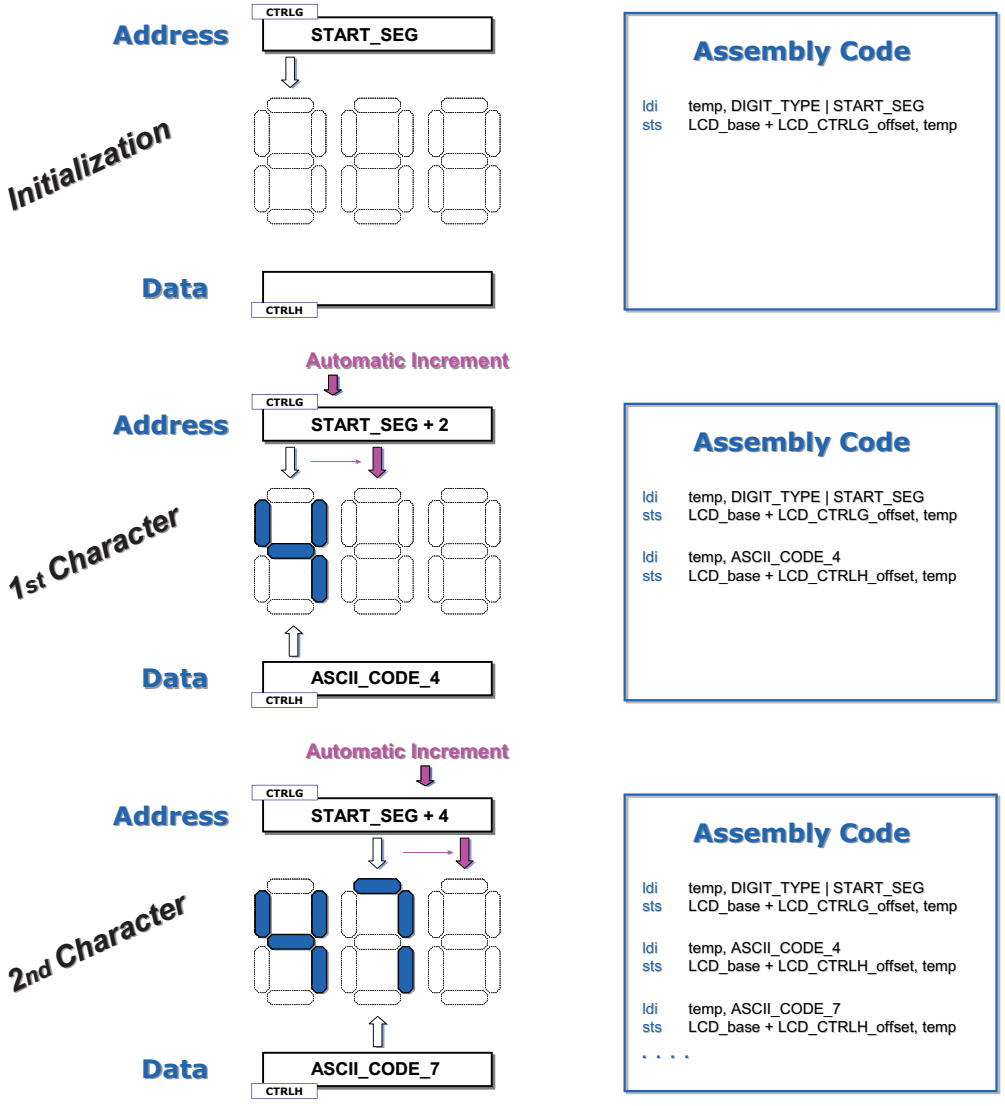


3 ASCII character mapping

3.1 Ease of use

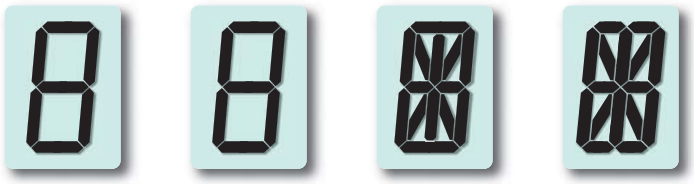
The Atmel ATxmega128B1 LCD controller can automatically handle ASCII characters. Instead of setting and clearing segments of a digit, the user enters the ASCII code and the digit decoder updates the segment values in the display memory.

Figure 3-1. ASCII mapping.



To initiate ASCII character mapping the user has to set-up the segment type and a pointer on the SEG line number where the group of digits begins (LCD_CTRLG). Then, this works as a FIFO. The user writes the ASCII value of the first digit into the control register (LCD_CTRLH). When the segments are updated, the pointer is automatically incremented (or decremented) with the number of SEG lines used for the digit. The pointer is now ready for a new ASCII writing.

Figure 3-2. Supported digit types.



7-Segment
- 3 COM term.
- 3 SEG term.

7-Segment
- 4 COM term.
- 2 SEG term.

14-Segment
- 4 COM term.
- 4 SEG term.

16-Segment
- 3 COM term.
- 6 SEG term.

3.2 LCD control register G

7	6	5	4	3	2	1	0	
TDG[1:0]		STSEG[5:0]						LCD_CTRLG

- **TDG[1:0]: Type of Digit**

This bit-field specifies the number of segments and COM/SEG lines of the digit.

TDG[1:0]	Type of digit
00	7-segment with COM[2:0] / 3 SEG
01	7-segment with COM[3:0] / 2 SEG
10	14-segment with COM[3:0] / 4 SEG
11	16-segment with COM[2:0] / 6 SEG

- **STSEG[5:0]: Start Segment**

STSEG bit-field defines the first segment line used to write the digit. This bit-field is automatically incremented or decremented (according to the DEC value of LCD.CTRLH register) by the number of SEG lines used in the digit.

TDG[1:0]	Type of digit	Increment/decrement number
00	7-segment - 3 COM / 3 SEG	3
01	7-segment - 4 COM/ 2 SEG	2
10	14-segment - 4 COM / 4 SEG	4
11	16-segment - 3 COM / 6 SEG	6

3.3 LCD control register H

7	6	5	4	3	2	1	0	
DEC	DCODE[6:0]							LCD_CTRLH

- **DEC: Decrement**

Writing this bit to one automatically decrements the STSEG bit-field of LCD_CTRLG register by the number of SEG lines used by the digit. If this bit is written to zero, the STSEG bit-field is incremented by the number of segment lines used by the digit. This action takes place once the digit decoding is finished and prepares the next call to the Digit Decoder.

- **DCODE[6:0]: Display Code**

DCODE bit-field will be computed by the digit decoder, and converted to display codes, and then automatically written into the display memory according to the STSEG value. The table entry code, DCODE [6:0], is the 7-bit ASCII code of the digit.

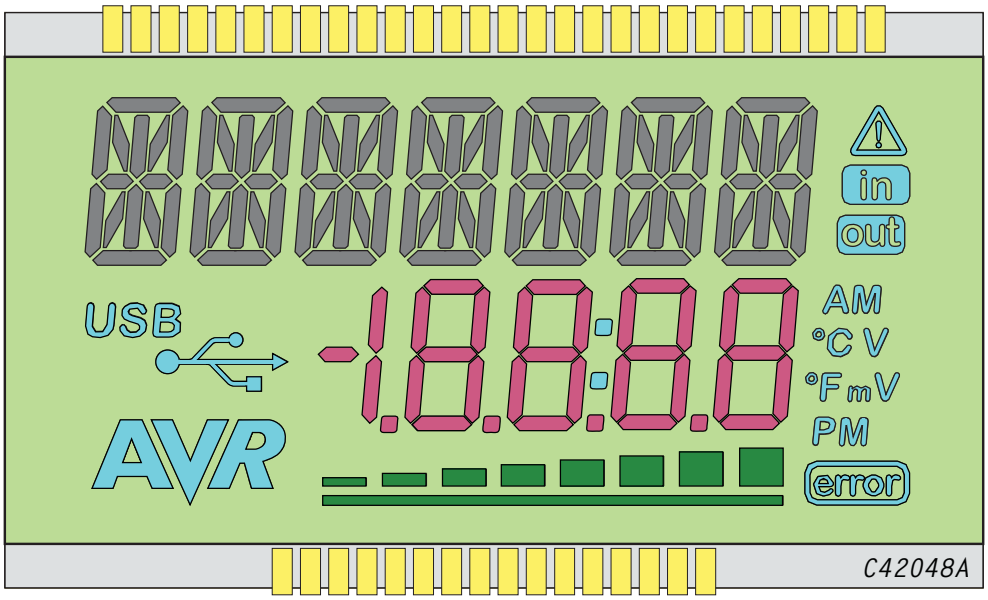
4 LCD C42048A glass

The LCD C42048A has been especially designed for the Atmel XMEGA-B1 Xplained kit. It comes with four COM by 40 SEG lines LCD glass running at 3.0V and it includes 4 groups of segments (pixels):

- Seven 14-segment alphanumeric digits
- Five 7-segment numeric digits with negative sign and DPs (decimal points)
- One bar graph of nine segments (pixels)
- 13 icons of one segment (pixel)

Total: 155 segments (pixels).

Figure 4-1. Segment groups of LCD C42048A.



4.1 14-segment alphanumeric digits

The alphanumeric digits have been routed to match with the third digit type (TDG[1:0]=2) supported by the Digit Decoder of the Atmel ATxmega128B1 device and defined in LCD_CTRLG register (14-segment with four COM lines).

This assignment leaves two pixels not used by the Digit Decoder in locations COM0/SEG(n+1) and COM3/SEG(n+2).

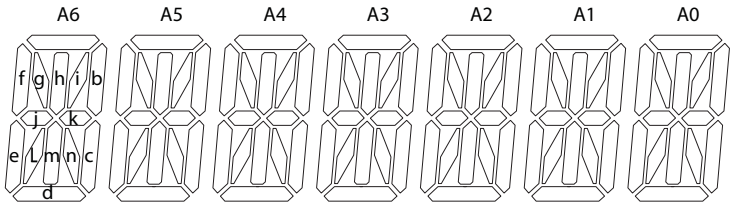
The alphanumeric digits are named from A0 up to A6.

- A6 (digit on the left) is driven by SEG[12:15]
- A5 is driven by SEG[16:19]
- …
- A0 (digit on the right) is driven by SEG[39:36]

NOTE

To keep the same character scan order than in a C-string (left to right), STSEG will have to be incremented (DEC=0) by 4.

Figure 4-2. Alphanumeric digit routing.



	SEG																			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
COM0													A6-h		A6-a	A6-g	A5-h		A5-a	A5-g
COM1													A6-i	A6-f	A6-b	A6-j	A5-i	A5-f	A5-b	A5-j
COM2													A6-k	A6-e	A6-c	A6-L	A5-k	A5-e	A5-c	A5-L
COM3													A6-n	A6-d		A6-m	A5-n	A5-d		A5-m

	SEG																			
	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
COM0	A4-h		A4-a	A4-g	A3-h		A3-a	A3-g	A2-h		A2-a	A2-g	A1-h		A1-a	A1-g	A0-h		A0-a	A0-g
COM1	A4-i	A4-f	A4-b	A4-j	A3-i	A3-f	A3-b	A3-j	A2-i	A2-f	A2-b	A2-j	A1-i	A1-f	A1-b	A1-j	A0-i	A0-f	A0-b	A0-j
COM2	A4-k	A4-e	A4-c	A4-L	A3-k	A3-e	A3-c	A3-L	A2-k	A2-e	A2-c	A2-L	A1-k	A1-e	A1-c	A1-L	A0-k	A0-e	A0-c	A0-L
COM3	A4-n	A4-d		A4-m	A3-n	A3-d		A3-m	A2-n	A2-d		A2-m	A1-n	A1-d		A1-m	A0-n	A0-d		A0-m

4.2 7-segment numeric digits

The numeric digits have been routed to match with the second digit type (TDG[1:0]=1) supported by the Digit Decoder of the Atmel ATxmega128B1 and defined in LCD_CTRLG register (7-segment with four COM lines).

This assignment leaves one pixel not used by the Digit Decoder in location COM3/SEG(n+1). This place has been used for DP and must be manually set/cleared.

The numeric digits are named from D0 up to D3.

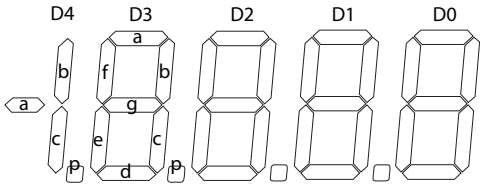
- D3 is driven by SEG[8:9]
- D2 is driven by SEG[6:7]
- …
- D0 (digit on the right) is driven by SEG[2:3]. D0 has no DP

NOTE

To keep the same character scan order than in a C-string (left to right), STSEG will have to be decremented (DEC=1) by 2.

D4 digit is specific and will only display four values: -, -1, 0 & 1. It will be manually driven. DP for D4 is located COM3/SEG10.

Figure 4-3. Numeric digit routing.



	SEG																			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
COM0			D0-a	D0-f	D1-a	D1-f	D2-a	D2-f	D3-a	D3-f	D4-a									
COM1			D0-b	D0-g	D1-b	D1-g	D2-b	D2-g	D3-b	D3-g	D4-b									
COM2			D0-c	D0-e	D1-c	D1-e	D2-c	D2-e	D3-c	D3-e	D4-c									
COM3				D0-d	D1-p	D1-d	D2-p	D2-d	D3-p	D3-d	D4-p									

	SEG																			
	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
COM0																				
COM1																				
COM2																				
COM3																				

5 Software implementation

5.1 Software driver

5.1.1 Code of driver

```
/**
 * \brief Send a sequence of ASCII characters to LCD glass.
 *
 * This function enables LCD segments (pixels) via the Digit Decoder.
 * The function will write the maximum number of bytes passed as
 * argument, and will stop writing if a 'NULL' character is found.
 *
 * \param  lcd_tdg    Type of digit to decode.
 * \param  first_seg  First SEG where the first data will be written.
 * \param  data       Data buffer.
 * \param  width      Maximum number of data.
 * \param  dir        Direction (==0: Left->Right, !=0: Left<-Right).
 */
void lcd_write_packet(enum LCD_TDG_enum lcd_tdg, uint8_t first_seg,
                      const uint8_t *data, size_t width, uint8_t dir)
{
    LCD.CTRLG = lcd_tdg | ((first_seg<<LCD_STSEG_gp) & LCD_STSEG_gm);
    if (dir != 0) {
        dir = LCD_DEC_bm;
    }
    while (width--) {
        if (*data == '\0') {
            break; // Stop on NULL char
        }
        LCD.CTRLH = dir | (*data++);
        }
    }
}
```

5.1.2 Driver arguments

- "lcd_tdg"
  This argument is used to choose one element of `LCD_TDG_enu` included in the device header ("iox128b1.h" if GCC, "ATxmega128B1.h" if IAR™):

```
typedef enum LCD_TDG_enum {
        LCD_TDG_7S_3C_gc  = (0x00 << 6), /* 7-segment  with 3 COMs */
        LCD_TDG_7S_4C_gc  = (0x01 << 6), /* 7-segment  with 4 COMs */
        LCD_TDG_14S_4C_gc = (0x02 << 6), /* 14-segment with 4 COMs */
        LCD_TDG_16S_3C_gc = (0x03 << 6), /* 16-segment with 3 COMs */
        } LCD_TDG_t
```

- "first_seg"
  Define the first SEG line of the first digit to write.
  Note that inside the digit, the scanning of the SEG lines is incremental but outside the digit, this means considering the set of digits, the scanning order is defined by the "dir" argument
- "*data"
  Pointer on the ASCII string to display
- "width"
  Number of digits of the line or maximum of digits to drive.
  Note that "`size_t`" is an ANSI type equal to an "`unsigned char`"
  In the driver, this value can be cut off if a null character is found (end of ASCII string marker)

- “dir”
  Define the direction of the digit scanning
  If this scanning is the same as in “`*data++`”, this argument must be equal to zero. If not this argument must be different of zero

5.1.3 Arguments for the 14-segment digits of C42048A

- “lcd_tdg”
  `LCD_TDG_14S_4C_gc` (14-segment with 4 COM lines)
- “first_seg”
  `12` (refer to Figure 4-2. Alphanumeric digit routing.) It corresponds to the first SEG line of the left digit A6
- “*data”
  `lcd_text` (name defined and used in the example)
- “width”
  `7` (refer to Figure 4-2. Alphanumeric digit routing.)
- “dir”
  `0` (refer to Figure 4-2. Alphanumeric digit routing.) A6 is the left digit, it will receive the character “`lcd_text[0]`”, first character pointed by “`*data++`”

5.1.4 Arguments for the 7-segment digits of C42048A

Because D4 is manually managed, D3 becomes the left digit for the driver.

- “lcd_tdg”
  LCD_TDG_7S_4C_gc (7-segment with 4 COM lines)
- “first_seg”
  `8` (refer to Figure 4-3. Numeric digit routing.) It corresponds to the first SEG line of the left digit D3
- “*data”
  `lcd_num` (name defined and used in the example)
- “width”
  `4` (refer to Figure 4-2. Alphanumeric digit routing.)
- “dir”
  `1` (refer to Figure 4-3. Numeric digit routing.) D3 is the left digit, it will receive the character “`lcd_numdata[0]`”, first character pointed by “`*data++`”

5.1.5 Control of limits

The Digit Decoder itself surveys the address limit. The automatic increment/decrement doesn’t overwrite the last addressed LCD data registers (LCD_DATAn) or adjacent registers.

5.2 Specialized macro functions for C42048A

5.2.1 Macro for the 14-segment digits

```
/**
 * \brief Write string to c42048a LCD glass alphanumeric field.
 *
 * This function will write the input string to the alphanumeric
 * field of the LCD glass.
 *
```

```
 * \param data Pointer to the data input string
 */
static inline void c42048a_write_alpha_packet(const uint8_t *data)
{
    lcd_write_packet(LCD_TDG_14S_4C_gc, FIRST_14SEG_4C, data, \
                     WIDTH_14SEG_4C, DIR_14SEG_4C);
}
```

Where:

Defined in device header:

- `LCD_TDG_14S_4C_gc` (see Section 5.1.2 Driver arguments)

Defined in glass component header:

- `#define FIRST_14SEG_4C 12`
- `#define WIDTH_14SEG_4C 7`
- `#define DIR_14SEG_4C 0`

5.2.2 Macro for the 7-segment digits

```
/**
 * \brief Write string to c42048a LCD glass numeric field.
 *
 * This function will write the input string to the numeric
 * field of the LCD glass.
 *
 * \param data Pointer to the data input string
 */
static inline void c42048a_write_num_packet(const uint8_t *data)
{
    lcd_write_packet(LCD_TDG_7S_4C_gc, FIRST_7SEG_4C, data, \
                     WIDTH_7SEG_4C, DIR_7SEG_4C);
}
```

Where:

Defined in device header:

- `LCD_TDG_7S_4C_gc` (see Section 5.1.2 Driver arguments)

Defined in glass component header:

- `#define FIRST_7SEG_4C 8`
- `#define WIDTH_7SEG_4C 4`
- `#define DIR_7SEG_4C 1`

6 Measurements

To measure the efficiency of the Digit Decoder built-in the Atmel ATxmega128B1 device, we will create a specific software driver to manually handle (setting or clearing) digit segments (pixels), directly writing to the LCD.DATAn registers.

This exercise is only done for 14-segment digit routed as defined for a digit type number 2 in the LCD.CTRLG register (TDG[1:0] = 2, LCD_TDG_14S_4C_gc) and for a direction “left to right” only.

Since the built-in Digit Decoder is not used, we will define our own ASCII table with 16-bit elements. The LSB nibble will be the set of ordered bits for COM0, and so, the MSB nibble will be the set of ordered bits for COM3. The index used to get digit segments (pixels) in this table will be the ASCII character code.

6.1 Code of manual driver

```
// 40 SEG lines in ATxmega128B1
#define   LCD_MAX_NBR_OF_SEG    40

// ASCII 14-segment table
const uint16_t ascii_table[] = {
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x2E74, 0x0440, 0x23C4, 0x2544, 0x05E0, 0x25A4, 0x27A4, 0x0444,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x2365, 0x07E4, 0xA545, 0x2224, 0xA445, 0x22A4, 0x02A4, 0x2724,
    0x0000, 0x0000, 0x0000, 0x0000, 0x2220, 0x0678, 0x1668, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x2660, 0x0000, 0x0000,
    0x1818, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
    0x0004, 0x07E4, 0xA545, 0x2224, 0xA445, 0x22A4, 0x02A4, 0x2724,
    0x0000, 0x0000, 0x0000, 0x0000, 0x2220, 0x0678, 0x1668, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x2660, 0x0000, 0x0000,
    0x1818, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000 };

/**
 * \brief Send a sequence of ASCII characters to LCD glass.
 *
 * This function enables and disables LCD segments (pixels)
 * to display ASCII 14-segment character.
 * The function will write the maximum number of byte passed as
 * argument, and will stop writing if a 'NULL' character is found.
 *
 * \param  lcd_tdg    Type of digit to decode.
 * \param  data       Data buffer.
 * \param  width      Maximum number of data.
 */
void lcd_pix_write_packet(uint8_t first_seg, const uint8_t *data,
                          uint8_t width)
{
register8_t *pix_reg;
uint8_t txt_i, com_i;
uint8_t       shift_todo, temp_data;

       // Loop on the Array of ASCII characters
       for (txt_i = 0; txt_i <    width; txt_i++) {
         if (*data == '\0') {
             break; // Stop on NULL char
         }
```

```
                    // Determine how many shift to do (0 or 4)
                    shift_todo = ((first_seg + (4 * txt_i)) % 8);
                    // Scanning all COM lines
                    for (com_i = 0; com_i < 4; com_i++) {
                  // Address of the corresponding    data register
                          pix_reg = (register8_t*) ((uint16_t) &LCD.DATA0)
                                + (com_i * ((LCD_MAX_NBR_OF_SEG + 7) / 8))
                                + ((first_seg + (4 * txt_i))/ 8);
                          // Get data
                          temp_data = (uint8_t)
                                ((ascii_table[data[txt_i]] &
                                (0x000F << (4 * com_i))) >> (4 * com_i));
                          temp_data <<= shift_todo;
                          // Writing in data register
                          *pix_reg = (*pix_reg & ~(0x0F << shift_todo)) |
temp_data;
                    }
              }
}
```

6.2 Automatic versus manual handling

6.2.1 Code size

Compiler: GCC
Compiler option: -Os

Automatic handling = 23% of manual handling

- Automatic handling: 42 bytes
- Manual handling: 180 bytes + 256 bytes per ASCII table

6.2.2 Execution time

Compiler: GCC
Compiler option: -Os
System frequency: 2 MHz
Number of digits ("`width`"): 7

Automatic handling = 30 times faster than manual handling

- Automatic handling: 59.705µs
- Manual handling: 1817.315µs

6.2.3 Execution time formulas

Unit: µC cycle
Function set-up: ST
Execution or set-up per digit: DG
Number of digits: n
Execution per COM: COM
Number of COM: y

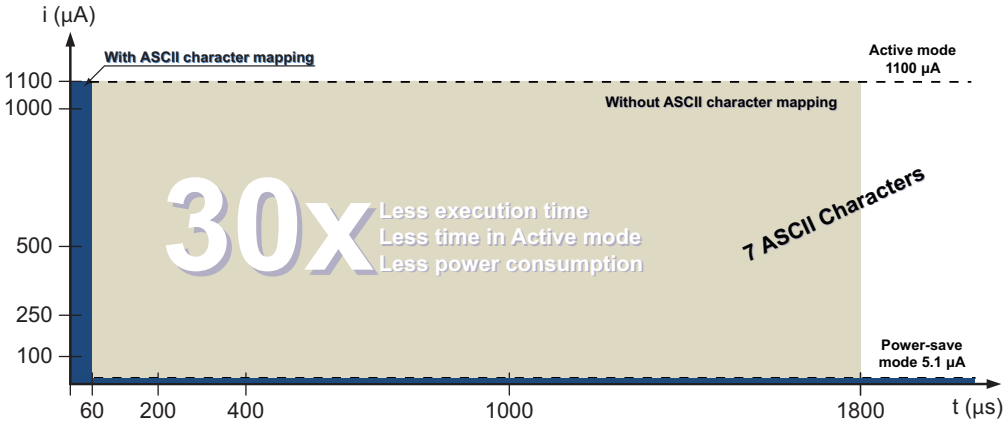
- Automatic handling: ST = 23, DG = 14, COM = 0
  ST + n DG
- Manual handling: ST = 89, DG = 139, COM = 92
  ST + n*DG + n*y*COM

6.2.4 Benefits of automatic handling

If there is no display update to do, the microcontroller can enter power-save mode. In active mode, the consumption typically is 1.1mA. In power-save mode, while LCD is running, the consumption typically is 5.1µA with the C42048A LCD glass connected.

- Less execution time
- Less time in active mode
- Less power consumption

Figure 6-1. Benefits of ASCII character mapping.



7 Recommended reading

It is recommended to read the following documents to get an overall idea about Atmel AVR XMEGA and especially about XMEGA B devices:

- XMEGA Manual and Datasheets
- AVR1000: Getting Started Writing C-code for XMEGA
- AVR1005: Getting started with XMEGA
- AVR1010: Minimizing the power consumption of XMEGA devices
- AVR1500: Xplain training - XMEGA Basics
- AVR1912: XMEGA-B1 Xplained Hardware User Guide
- AVR 1926: XMEGA-B1 Xplained Getting Started

8 Table of contents

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Milennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan
16F, Shin Osaki Kangyo Bldg.
1-6-4 Osaki Shinagawa-ku
Tokyo 104-0032
JAPAN
Tel: (+81) 3-6417-0300
Fax: (+81) 3-6417-0370

8451A-AVR-11/11