Proto-Counter Software

The Proto-Counter software is written in C++. It can be installed as an Arduino library. Download the folder „ProtoCounter“ from the Proto-Counter Github repository and in the Arduino IDE go to „Sketch → Include Library → Add .ZIP Library…“.

The class „ProtoCounter“ takes care of all basic tasks like:

- multiplexing the display
- displaying numbers and text
- de-bouncing of the push buttons
- control of external shift registers (optional)
- evaluation of an a potentiometer (optional)

The software uses less than half of the flash memory of the microcontroller leaving plenty of room for your own ideas.

Program Examples

The folder „examples“ contains some example code.

EventCounter

This example implements a simple event counter with two inputs. It counts high-to-low transitions at the inputs. Input A always increments the counter value by one. Input B can be configured to either increment or decrement the counter. When the counter value reaches a set limit an output becomes active for a certain time. For both inputs the program defines a dead time, which prevents spurious countings if you use mechanical switches as inputs.

TeaTimer

TeaTimer illustrates how a timer with adjustable time can be realized. The operation is described in the program. The code has already been prepared to enable a device while the timer is running, or trigger an alarm when the timer expires.

ProtoCounter Test

The program „ProtoCounter_Test.ino“ shows the usage of ProtoCounter functions. You can use it as a functional test after assembling the kit. Further description can be found in the program code.

Configuration

There are some parameters that can be configured in the headerfile `ProtoCounter.h` but this makes sense only for special cases. The meaning of the different parameters is explained in the file. You can specify:

- pin assingment
- number of external input shift register bits
- number of external output shift register bits
- disabling the analog knob
- de-bouncing time for the push buttons
- duration of a long button press

For disabling the analog knob out-comment the corresponding line.

```
//#define ANALOG_ENABLE
```

The external shift registers are disabled by defining their bit count as zero.

```
#define SH_REG_IN_BITCOUNT    0
#define SH_REG_OUT_BITCOUNT   0
```

Interrupt Control

Under Arduino the ProtoCounter library uses compare B interrupt of timer0. This interrupt periodically triggers the call of method `update()`. Complete control of the ProtoCounter therefore happens automatically in the background and the 16-bit timer (timer1) can be used by your application without restrictions.

The rate at which `update()` is called depends on the cpu speed and is F_CPU / 16384.

Programming without the Arduino environment

`update()` is the fundamental method which is responsible for all the inner workings of the Proto-Counter. If you don't use the Arduino environment you have to call `update()` periodically. This is conveniently done by a timer interrupt that is triggered e. g. every millisecond.

```
/* Main.cpp */
#include <avr/interrupt.h>
#include "ProtoCounter.h"

// system timing
#define SYS_TIMER_FREQ    1000    // system timer frequency (Hz)
#define SYS_CYCLE (uint8_t)(0.5 + F_CPU / (64.0 * (double)SYS_TIMER_FREQ))
...
```

```
int main(void)
{
    // setup timer0 which is used as system time base
    OCR0A  = SYS_CYCLE;        // set dot matrix refresh time
    TCCR0A = 0;           // normal mode
    TCCR0B = (3 << CS00);   // set prescaler
    TIMSK  = (1 << OCIE0A); // enable timer interrupt
    sei();            // enable interrupts to start the ProtoCounter engine
    ...
}

ISR(TIMER0_COMPA_vect)
// system timer interrupt
{
    OCR0A += SYS_CYCLE;        // setup next interrupt cycle
    sei();            // re-enable interrupts
    ProtoCounter::update();    // run the ProtoCounter engine
}
```

Proto-Counter Methods

In the following we assume that a ProtoCounter object has been instantiated.

```
#include "ProtoCounter.h"

ProtoCounter pc;
```

init()

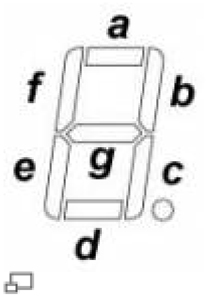
Initialize I/O pins and clear the display. Member variables `decimal_places` and `dimming` are set to their default values. You may change their values as required.

```
pc.decimal_places = 2;
pc.dimming = 0;
```

clearDisplay()

Clear the 7-segment-display (all segments off).

getDisplay(pos)



Display segments

Return the display content of the specified digit. pos = 0 is the least significant digit (to the very right). Bits 0 - 6 correspond to display segments a - g.

setDisplay(led_pattern, pos)

Writes the led pattern to the specified digit. pos = 0 is the least significant digit (to the very right). Bits 0 - 6 correspond to display segments a - g.

writeChar(ascii_code, pos)

Writes an ASCII character to the specified digit. Also see documentation of the character font.

writeString_P(str_pointer)

Writes a string to the display using the character font. The string resides in flash memory and must be terminated with a null character.

```
#include <avr/pgmspace.h>

pc.writeString_P(PSTR("ABC"));
```

writeInt(val)

Display a signed integer (16 bit). Allowed range is -99 to +999 (MIN_DECIMAL bis MAX_DECIMAL). When exceeding this range the display shows „UFL“ (underflow) or „OFL“ (overflow).

writeHex(val)

Display an 8-bit value as a hexadecimal number. The number is postfixed with „h“ to indicate that it is hexadecimal.

setDimming(dim)

Lower the brightness of the display which also reduces the current consumption of the ProtoCounter.

Reasonable values for the dimming parameter are from 0 (maximum brightness) to 8.

setDecimalPlaces(decimals)

Defines the number of decimal places in case jumper J1 has been set to turn on the decimal point.

setAnalogResolution(ana_res)

To avoid a jitter of the analog knob values one can reduce its resolution. The following constants are predefined.

```
#define ANALOG_MAX_RESOLUTION 0
#define ANALOG_129_DETENT_STEPS   1
#define ANALOG_86_DETENT_STEPS    2
#define ANALOG_65_DETENT_STEPS    3
#define ANALOG_43_DETENT_STEPS    4
#define ANALOG_33_DETENT_STEPS    5
#define ANALOG_22_DETENT_STEPS    6
#define ANALOG_OFF        7
```

getButton()

Return the state of the two push buttons. You can react to it using the predefined events in ProtoCounter.h.

```
/* ProtoCounter.h */
// push button events (do not change)
#define BTN1_PRESSED      (BUTTON1 | PB_PRESS)
#define BTN1_RELEASED     (BUTTON1 | PB_RELEASE)
#define BTN1_LONGPRESSED  (BUTTON1 | PB_LONGPRESS)
#define BTN2_PRESSED      (BUTTON2 | PB_PRESS)
#define BTN2_RELEASED     (BUTTON2 | PB_RELEASE)
#define BTN2_LONGPRESSED  (BUTTON2 | PB_LONGPRESS)


/* Main.cpp */
int main(void)
{
    ...
    if (pc.getButton() == BTN1_RELEASED) {
        pc.buttonAck();
        /* code to react on push button */
    }
    ...
}
```

buttonAck()

Acknowledge the press of a push button by clearing the button.

readShiftRegister()

Connected shift registers are updated with every call of `update()` (see Proto-Counter Engine). With readShiftRegister you get the state of the inputs at the time of the last update. The returned data type depends on the number of shift register bits (parameter SH_REG_IN_BITCOUNT) and can be byte, word or longword.

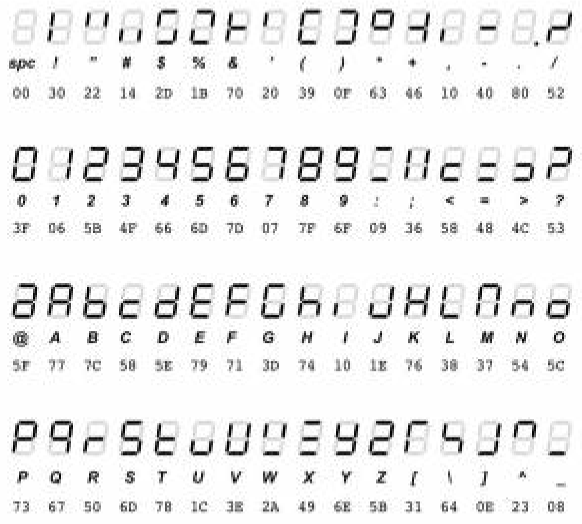
writeShiftRegister(data)

The data will be shifted out during the next update cycle. The data type depends on the number of shift register bits (parameter SH_REG_OUT_BITCOUNT) and can be byte, word or longword.

getAnalog()

Return a value (uint8) that corresponds to the position of the analog knob.

Character Font



spc	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
00	30	22	14	2D	1B	70	20	39	0F	63	46	10	40	80	52
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
3F	06	5B	4F	66	6D	7D	07	7F	6F	09	36	58	48	4C	53
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5F	77	7C	58	5E	79	71	3D	74	10	1E	76	38	37	54	5C
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
73	67	50	6D	78	1C	3E	2A	49	6E	5B	31	64	0E	23	08

7-segment character font

For displaying text a 7-segment character font is used. It contains ASCII-characters chr(32) to chr(96). Character codes 0 to 15 are mapped to the hexadecimal digits '0' to 'F'. Codes 16 to 31 and above 127 are replaced by a space character (all segments off).

If you like to create your own character font this online tool might come in handy (http://www.uize.com/examples/seven-segment-display.html).

ProtoCounter, electronics, kit, programming