

**EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
ORGANIZATION EUROPEENNE POUR LA RECHERCHE NUCLEARE**

CERN — PS DIVISION

PS/BD/Note 2002-162

BURST GENERATOR

Michal Ostapowicz

Abstract

The Burst Generator is a NIM module designed to generate bursts of fast pulses. It can be programmed to generate a given number of pulses after a programmable delay following a trigger pulse. The frequency of those pulses can be either 250MHz or a few values around 500MHz. The module will be used to generate clock impulses for an Acqiris data acquisition module.

This paper describes how to tune, test and operate the Burst Generator.

Geneva, Switzerland
September 26, 2002

1. Introduction

The purpose of the Burst Generator is to allow acquisition of selected parts of a signal, and thus use the digitizers memory more efficiently. The Burst Generator was designed for use with the Acqiris Cougan 500-4 digitizers and takes advantage of the latter's external clock input.

The Burst Generator is a NIM module designed to generate bursts of fast pulses. It can be programmed to generate a given number of pulses after a programmable delay following a trigger pulse. The frequency of these pulses can be either 250MHz or a few values around 500MHz. The burst is used to provide the external clock for the Acqiris module.

2. Functional description

Referring to the block diagram (fig. 1), the Burst Generator consists of 3 sub-circuits:

1. "slow" TTL logic, used as an interface to a PC-compatible parallel port,
2. a PLL, which generates a stable 500MHz (or 250MHz) clock,
3. fast ECL logic, used to gate the clock produced by the PLL. This part is represented by two blocks (*delay* and *length*).

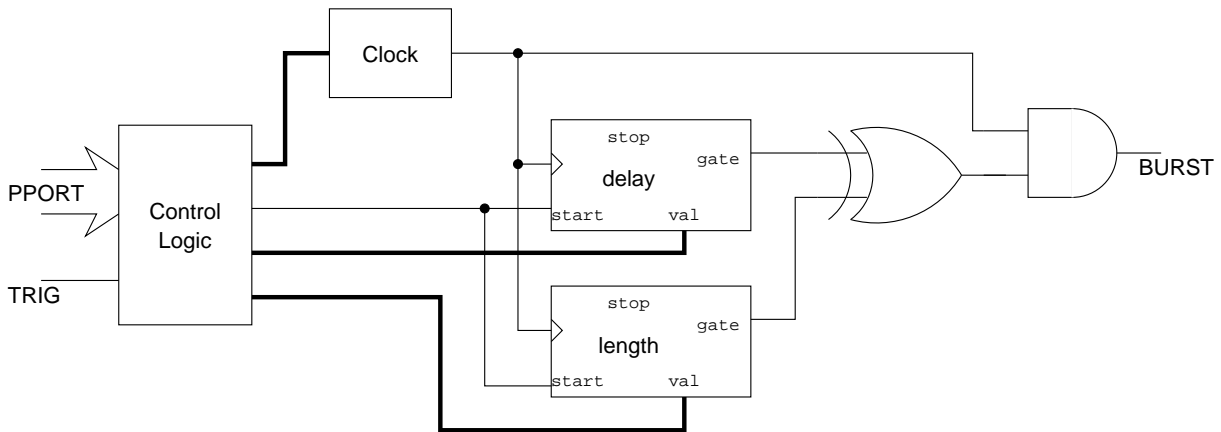


Figure 1: Block diagram of the Burst Generator

For the full schematic diagram consult the EDMS web page [4].

2.1. Control Logic

The control logic is realized using an Altera [1] chip. It provides an interface to a PC-compatible parallel port. The timing of the operation is shown on figure 2. The Burst Generator accepts 5 bytes of data, which are dispatched to consecutive internal registers by a 5-state counter that is automatically reset 45ms ($\pm 20\%$) after the last transferred byte.

The control logic is fast enough to keep up with parallel port activity without any delays on the computer side. Preferably the standard operating system parallel port driver should be used to communicate with the module. In case of a self-made driver, table 1 sums up the required timings.

The meaning of data sent to the module is described in figure 3. The `Ctrl` word has more than one function: bit 6 enables trigger, bit 7 enables 250MHz mode, and bits 5...0 are used to control the frequency (as described in section 2.2.).

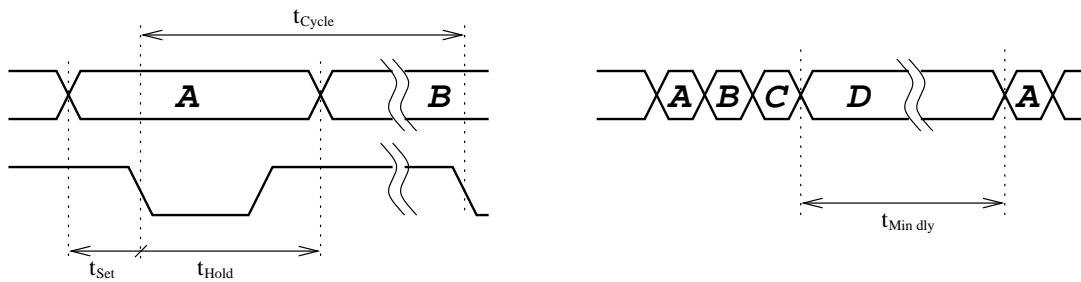


Figure 2: Writing cycle

Table 1: Timing Parameters

Parameter	Description	Value	Unit
t_{Set}	set-up time	20	ns
t_{Hold}	hold time	18	ns
t_{Cycle}	maximum duration of writing cycle	30	ms
t_{Mindly}	minimum delay after uncompleted programming cycle	60	ms

2.2. Clock generation (PLL)

A PLL with a 25MHz quartz reference generates the fast clock. The loop was designed to be very stable, and thus it is very slow (it need not be fast). An important fact to keep in mind, while interacting with the Burst Generator, is that setting a new working frequency takes about a second.

The frequency is set by loading a frequency divider value into the PLL (the 6 least significant bits of the `Ctrl` word). To calculate the output frequency one can use the formula:

$$f = (V + 2) \cdot 25\text{MHz},$$

where V is the value preloaded into the divider.

Because of a limitation of the VCO used in the PLL, only 4 steps are possible: 500, 475, 450 and 425MHz.

Setting bit 7 of the `Ctrl` word divides the frequency by 2. In this case only the generation of 250MHz is possible — the *Clock frequency modifier* should be set to `0x12`.

2.3. Counters

There are two identical counters: *delay* and *length*. Each of them is basically a count-down counter preloaded on the triggering event. The triggering event must be enabled by setting bit 6 of the `Ctrl` word.

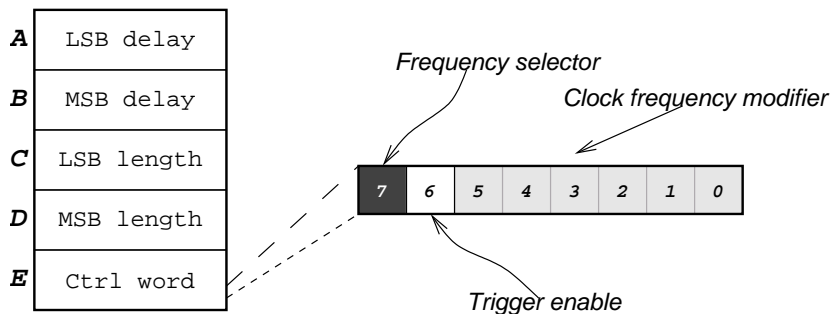


Figure 3: Programming sequence

The circuit is symmetrical. By convention, the first counter is called “delay”, the second “length”. Each counter counts down and then stops. Output pulses are produced while one counter is stopped and the other is still running. The number of pulses so produced is therefore simply the difference in the preset values.

For technical reasons one should add 64 to the required values before loading them into the counters. For detailed description see section 4.4..

3. Usage of the Burst Generator

3.1. Front panel

On the front panel of the Burst Generator there are 3 connectors:

1. TRIG: LEMO connector for the triggering signal. It accepts TTL levels, falling edge active.
2. BURST: BNC connector for the output of the module. It is DC coupled, 1600mV pulses into 50Ω to GND.
3. PPORT: Parallel port connector.

There are also 3 LEDs:

1. TRIG: blinks on trigger. Note that the module is triggered only if the trigger enable bit is set.
2. PPORT: blinks on the falling edge of the `strobe` signal — parallel port activity.
3. ERROR: is turned on if:
 - the PLL is not locked,
 - the *Clock frequency modifier* has an inappropriate value.

3.2. Parameters of Burst Generator

The main parameters of the Burst Generator are shown in table 2.

Table 2: Run-time parameters

<i>Parameter</i>	<i>Value</i>	<i>Unit</i>	<i>Note</i>
Min. delay (500MHz)	12	ns	<i>delay</i> = 64 (preset value)
Min. delay (250MHz)	22	ns	<i>delay</i> = 64 (preset value)
Max. delay (500MHz)	8062	ns	only one pulse generated
Max. delay (250MHz)	16126	ns	only one pulse generated
Min. number of pulses	1		
Max. number of pulses	$4095 - \textit{delay}$		<i>delay</i> : preset value

The external trigger is synchronized to the internal clock. Then, the delay, measured from the synchronized trigger is $5 \text{ clock cycles} + 2\text{ns}$.

Figure 4 shows a screen dump from the LC584A scope. The Burst Generator has been set to produce 4 pulses. The wiggles following the burst are reflections from the scope input, re-reflected from the Burst Generator output. The output impedance of the module is 150Ω , and the load should match the cable to avoid reflections.

3.3. Example program

The example program sends the following configuration to the module:

- clock frequency $f = 500\text{MHz}$ without division,
- delay: 4000 clock cycles (*i.e.* $8\mu\text{s}$),

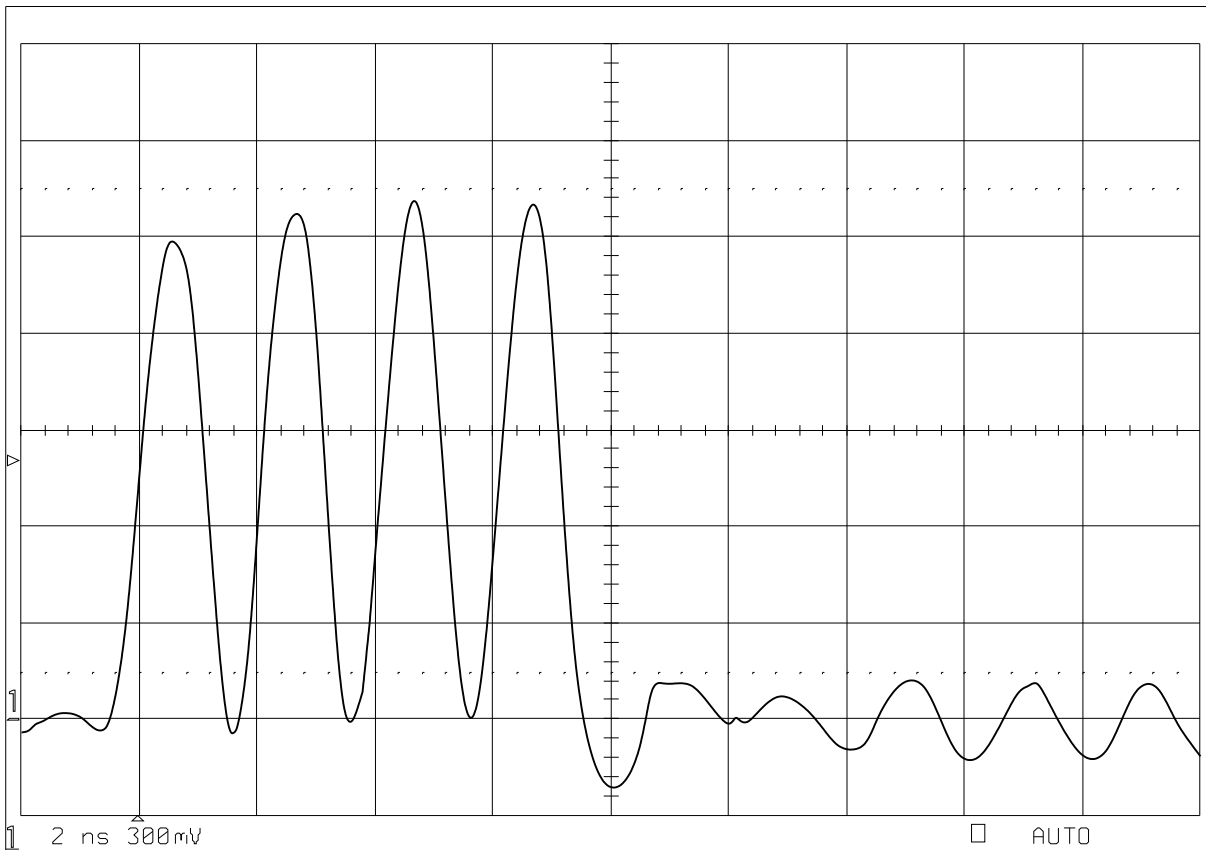


Figure 4: Oscilloscope of the Burst Generator output signal

- length: 4005 clock cycles (*i.e.* the module will generate 5 impulses),
- enable external trigger.

If this operation fails the program displays an appropriate error message and returns with an error code (-1).

The program needs write access to the first parallel port (/dev/lp0 — the module must be connected to this port), or the superuser privileges.

```

/*
 * send_bg_config()    sends configuration to the Burst Generator using
 *                    port /dev/lp0
 *
 *      delay, length  in clock cycles
 *
 *      trig, div      boolean (enable/disable) trigger, frequency division
 *
 *      freq           frequency (in MHz)
 */
static int send_bg_config(unsigned int delay, unsigned int length,
                          int trig, int div, int freq)
{
    int fd;
    unsigned char tab[5];

    delay += 64;                               /* correction for ``64 problem`` */

```

```

length += 64;

tab[0] = delay          & 0xff; /* LSB delay */
tab[1] = (delay >> 8)  & 0x0f; /* MSB delay */
tab[2] = length        & 0xff; /* LSB length */
tab[3] = (length >> 8) & 0x0f; /* MSB length */
tab[4] = freq / 25 - 2; /* Clock frequency modifier */
tab[4] |= trig ? 0x40 : 0; /* Trigger enable */
tab[4] |= div  ? 0x80 : 0; /* Frequency selector */

if ((fd = open("/dev/lp0", O_WRONLY)) == -1)
    return -1;

if (write(fd, tab, 5) == -1) {
    close(fd);
    return -1;
}
close(fd);

return 0;
}

int main(void)
{
    if (send_bg_config(4000, 4005, 1, 0, 500) == -1) {
        perror("couldn't send configuration to the module");
        return -1;
    }

    return 0;
}

```

The `send_bg_config()` function prepares an array of 5 bytes and sends it to the parallel port. It works with Linux on i386. For other operating systems only the “sending” part will differ.

4. Detailed circuit description

4.1. Clock generation — PLL

To generate 500MHz, a PLL is used with a reference of 25MHz. To divide the VCO frequency an MC10E136 counter is used. Although it can count up to 550MHz, it cannot divide at this speed — the setup and hold time specifications are not met.

To overcome this problem an extra D-flip-flop is used in the “preset” signal path (see figure 5). This implies that the preload value for this divider is $V = N - 2$, e.g. 18, to divide by 20.

The 250MHz is simply taken from the Q_0 pin of the counter. This means that only even division ratios will yield a usable clock. To simplify error detection only a *frequency modifier* of $0x12$ is allowed, when the Q_0 output is used. However, it would work with $0x10$ (225MHz) as well.

4.2. PLL Tuning

To set up the PLL one must take the following steps.

1. Send the programming sequence for the highest frequency — in this case only the last byte matters.
2. Adjust inductance L_3 (see fig. 5) so that V_{TUNE} is about 8V. The ERROR LED should turn off to indicate that the PLL is locked.

3. Send the programming sequence for the lowest frequency (4 steps lower — steps are 25MHz each).
4. If the PLL is not locked (ERROR LED lights up) go back to point 2, but set V_{TUNE} a bit higher.

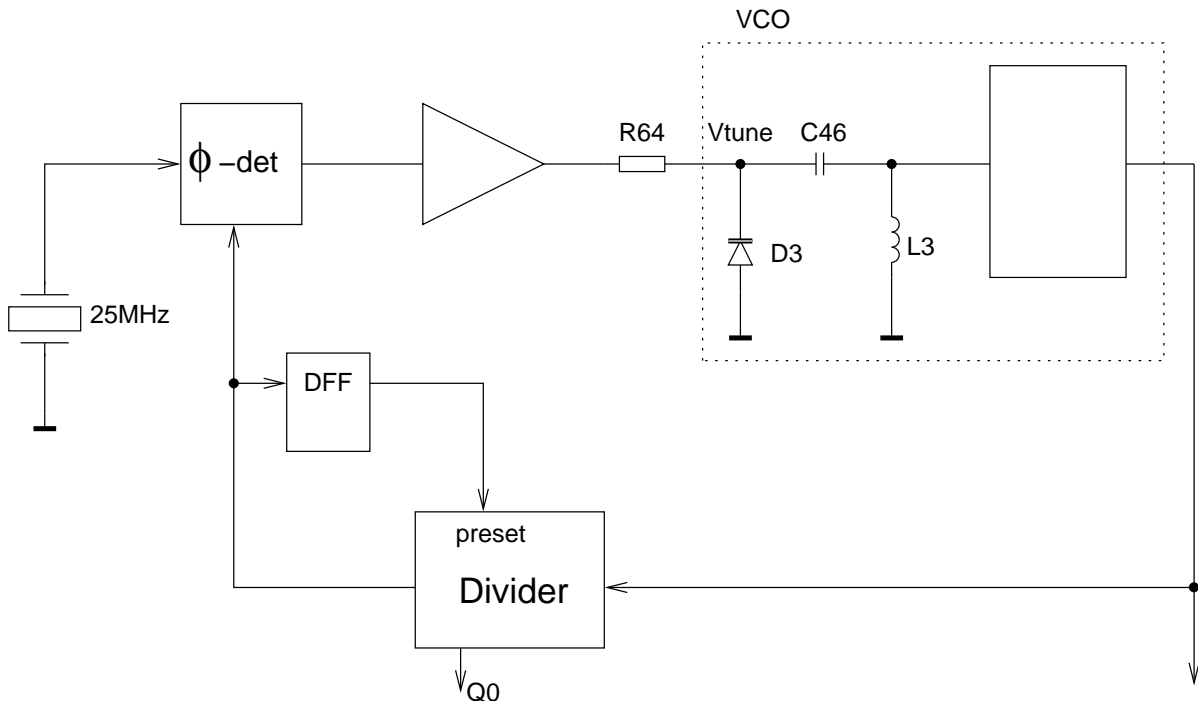


Figure 5: Simplified schematic diagram of the PLL

4.3. Testing

To test the module, its symmetry can be put to good use. One can set the difference in the *length* and the *delay* counters to a constant value x . Then the preloaded counter values can be changed by one at a time. The number of output pulses should stay the same and they should be one clock later with respect to the trigger every additional step.

The same operation should be repeated with the function of the counters reversed (*i.e.* if in the previous step *length* was lower — now *delay* should take its place). The difference between the counters should remain constant.

4.4. 12 bit presetable counter

The *delay* and *length* counters are made of two Motorola MC10E136 counters. The simplified schematic diagram is shown in figure 6. The counter should count down from a preloaded value and then stop. When stopped it should produce a signal used to control the gate.

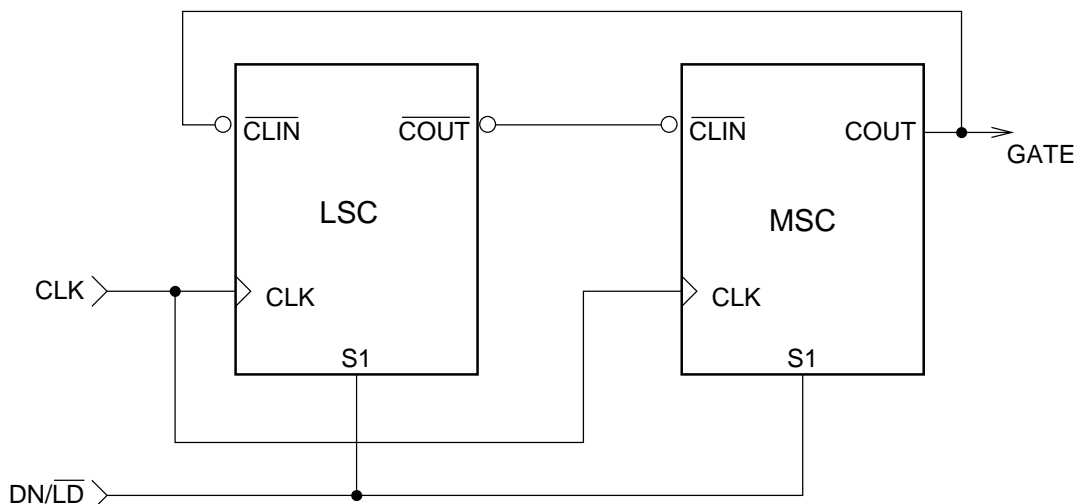


Figure 6: Simplified schematic diagram of the delay or length counter

There are 3 ways of “holding” (stopping) an MC10E136 counter.

- Using S1, S2 combination: this would require additional logic to be used, and since 500MHz is almost at the limit of the MC10E136’s capabilities this would cause timing problems.
- Using \overline{CIN} . This presents another problem: to produce a “hold signal” one can use a zero detector (additional logic) or \overline{COUT} . When the counter is held by \overline{CIN} , the \overline{COUT} goes high, removing the “hold signal”.
- Using \overline{CLIN} . This way the counter is held, but “not completely”, in the sense that \overline{COUT} and \overline{CLOUT} can still change, even though $Q_0 \dots Q_5$ stay put.

In the implementation the last possibility was used. It works as follows: (the timing diagram is shown in figures 7 and 8)

1. Preset: when S1 goes low, the counters are loaded on the next CLK (the state of the carry outputs does not matter). At the same time \overline{CIN} and \overline{CLIN} go high.
2. Despite the fact that \overline{CLIN} of the LSC is low, the counter is still in the hold state — the \overline{CLIN} inputs work with one clock cycle of delay. The counter, however, can calculate \overline{COUT} properly. In figure 7 it stays high, but in figure 9, where the counter was preloaded with the terminal state (0), it goes low.
3. The LSC starts counting down until it reaches 0 — then \overline{COUT} goes low.
4. Since the MSC uses \overline{CLIN} to hold, it is taken into account on the *next* clock. When this happens the gate signal is produced and so is the hold signal for the LSC.
5. The LSC finally stops after yet another clock at -3, but it does not affect the gate signal.

As we can see on figure 9 the LSC works in the same way when set to 0. As was mentioned before, the minimum value that can be preset is 64 (*i.e.* 1 to the MSC). The reason for this is the fact that counting stops a few clock cycles after the MSC reaches zero irrespective of the state of the LSC.

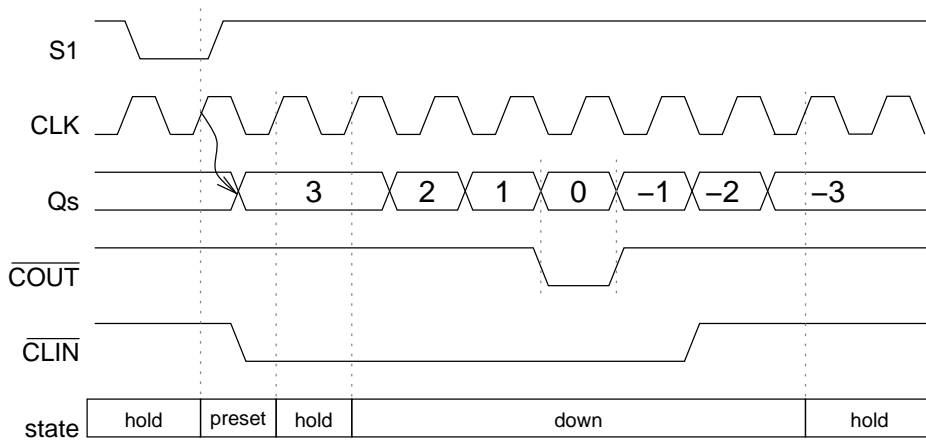


Figure 7: LSC (preloaded: 3)

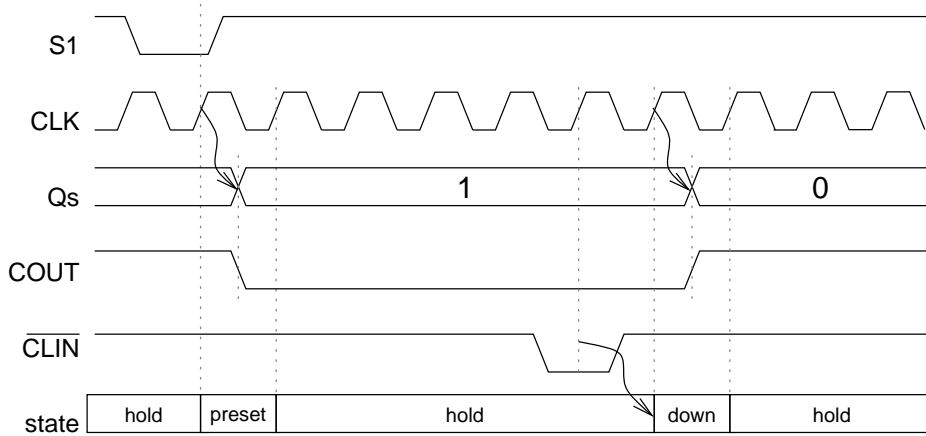


Figure 8: MSC (preloaded: 3)

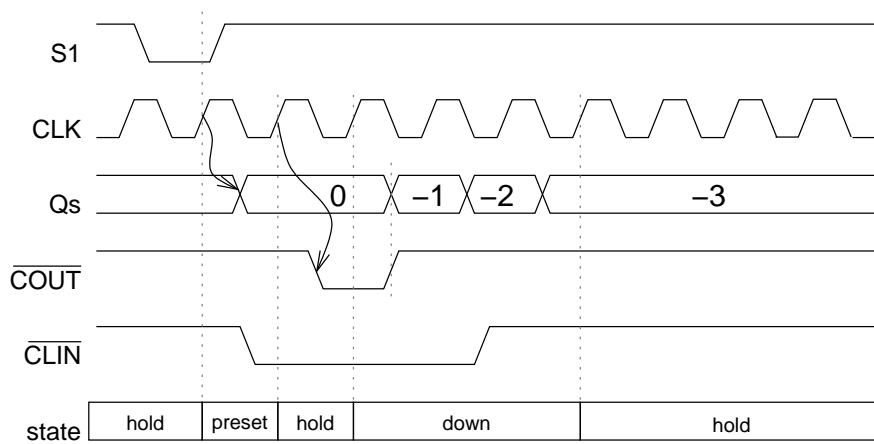


Figure 9: LSC (preloaded: 0)

5. Conclusion

The Burst Generator provides an efficient means to optimize the memory use of digitizers by sampling only interesting parts of the input signal. Although the Burst Generator was designed especially to work with Acqiris Digitizers, it can be used (maybe after adjusting the output voltage) as a more general tool. It should be noted however that quite a lot of collaboration with engineers from Acqiris was needed to achieve compatibility.

Acknowledgements

I would like to thank Maria Elena Angoletta, Jeroen Belleman and Jose Luis Gonzalez from for their help in realizing this project.

References

- [1] Altera: *Device Data Book*, Altera Corporation 1999
- [2] Motorola: *High Performance ECL Data ECLinPS and ECLinPS Lite*, Motorola, Inc. 1993
- [3] Acqiris: *Acqiris Digitizers User Manual*, March 2001.
- [4] Burst Generator: <http://edmsoraweb.cern.ch:8001/cedar/item.search> the *Item number* to search for is CERN-0000008434.
- [5] Web page: <http://cern.ch/jeroen/burstgen>