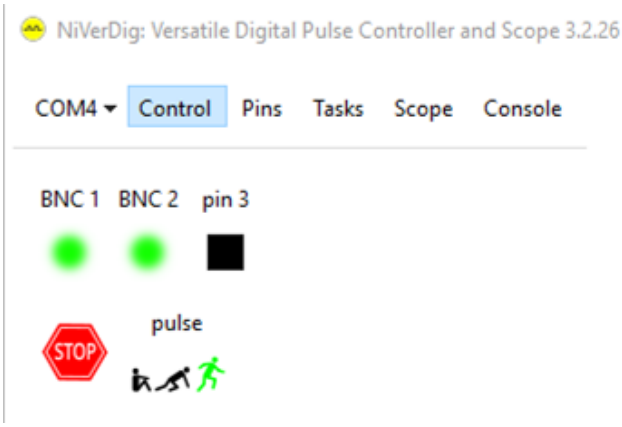
NiVerDig: Arduino-based Versatile Digital Controller and Scope

Introduction

NiVerDig is a versatile digital signal controller scope based on an Arduino Uno or Mega board. The NiVerDig Arduino Sketch allows flexible configuration of tasks that can be started and stopped by serial command or external digital input signals. When programmed, the NiVerDig can execute the tasks autonomously. When connected the PC the unit can be configured dynamically. The Scope mode allows recording the digital events to PC. A wxWidgets windows control program is available as a GUI front-end.

Control Panel



After selecting the COM port of the connected device, the Control Panel becomes active. If the connection fails, select 'Upload NiVerDig Sketch to Uno/Mega' to program the Arduino with the sketch. The Control panel shows all defined Pins on the top row and all defined Tasks on the bottom row.

pins

The pin color tells the current state: green if high (5V), black if low (0V). Input pins have a round icon, output pins have a rectangular icon. The state of the output pins is toggled upon click with the mouse.

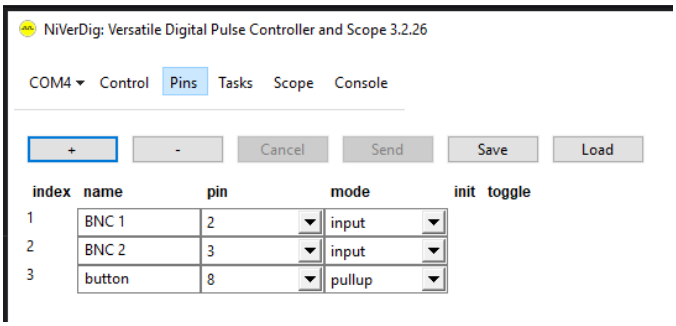
stop

The first icon on the task row is a red STOP button. Click on this button to halt all task. Click again to arm/start the tasks again. Note: if the NiVerDig has a physical button, hold this button for 1 second during power-up to halt all tasks and prevent starting any tasks automatically.

tasks

Tasks can be 'idle' (sitting icon), 'armed' ('on your mark' icon) or 'running' (running icon). The colored icon reflects the current state of each task. Click on the idle, arm or run icon to change the state of the task. An 'idle' task will not be started by a start trigger. After 'Arm'ing the task, a start trigger will 'start' it. On completion of the task, it will become 'idle'. The options 'arm-on-startup' and 'arm-on-finish' can be set to arm the task automatically on boot or task completion.

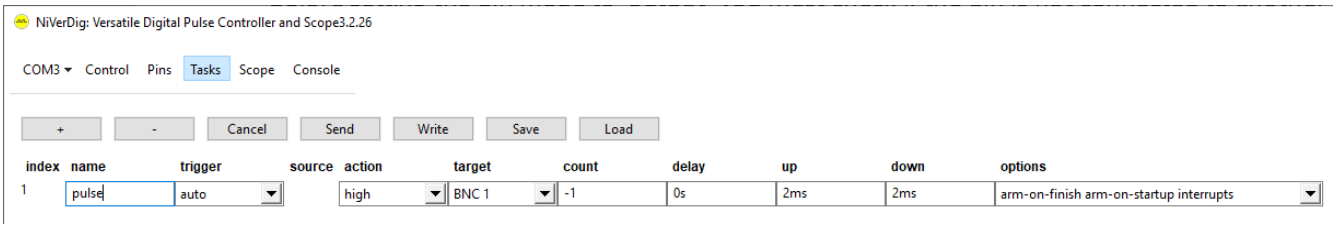
Pins



The Pins panel shows the definition of the NiVerDig logical pins. For each pin a 'name' is defined and the actual hardware pin they refer to. Pins can be in the input or output mode. The 'Pullup' mode is an input mode in which the pin voltage is pulled up to 5V. Short cutting the pin to ground e.g. with a button will pull it down. For the 'output' mode, you must specify the initial state of the output pin.
Before setting the pin mode to 'output' you must check if the pin is not connected to another output, button or shortcut ted to ground because the shortcut current can destroy the Arduino output port. The + and – buttons can be used to increase or decrease the number of logical pins. After changing the pin configuration, press 'Send' to apply the changes to the device. Pressing the 'Save' and 'Load' buttons save and load the pin configuration to text file. Tasks referring to the pins are not updated automatically for the changes in the pin definition. After any change, validate the definition of the tasks.

Tasks

The tasks panel show all defined tasks. The + and – buttons can be used to change the number of tasks. 'Send' sends the task definition to the device. By default 'Send' will only update the current task definition in memory. Press the 'Write' button to save it to EEPROM.



The fields accept the following values:

field	**values**
trigger	determines when the task is started
	auto: start the task automatically
	manual: start the task from software or by another task
	up: start the task when the source pin goes from low to high
	down: start the task when the source pin goes from high to low
	any: start the task when the source pin level changes
	high: run the task as long as the source pin is high
	low: run the task as long as the source pin is low
	start: start the task if the source task starts
	stop: start the task if the source task stops
source	source pin or task (depending on trigger)

action	sets the task activity. high: a digital pulse on the destination pin starting high low: a digital pulse on the destination pin starting low toggle: toggle the destination pin state arm: arm the destination task start: start the destination task restart: restart the destination task kick: start the destination task if idle, stop if running. stop: stop the destination task
target	destination pin or task (depending on the action)
count	number of iterations. One iteration consists of an 'up' action and a 'down' action. Specify a 'count' of 0 to execute only the 'up' action. Specify a 'count' of -1 for a continuous task.
delay	the period between the trigger and the first 'up' action. Without unit the delay is in microseconds. Add the 'ms' or 's' unit for milliseconds or seconds.
up	the period between the 'up' and 'down' actions. Without unit the delay is in microseconds. Add the 'ms' or 's' unit for milliseconds or seconds.
down	the period between the 'down' and 'up' actions. Without unit the delay is in microseconds. Add the 'ms' or 's' unit for milliseconds or seconds.
options	the options for this task: arm-on-startup: arm the task automatically when the device boots arm-on-finish: arm the task automatically when the task ends interrupts: start and tick the task using hardware interrupts

Note: if a task that is started automatically is written to EEPROM that exceeds the Arduino speed capabilities, the NiVerDig will be unresponsive after boot. In that case hold the button for 1 seconds on boot to halt all tasks and update them. If that does not help, keep the button pressed for 5 seconds to reset all pin and task definitions to factory default.

interrupts

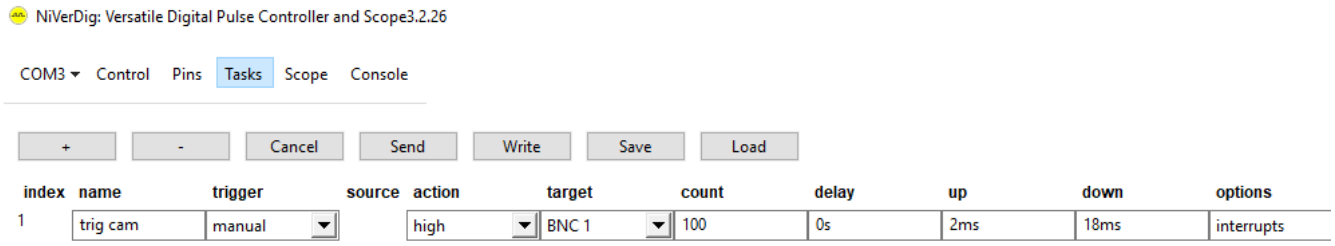
The AVR chip is a single thread device. The main thread runs in a loop and checks the pins and tasks if action is required. The time for one loop is about 300 us. For this thread, the jitter (fault in timing) is about 300 to 600 us. Independently of the main thread, the chip features a thread that runs upon a hardware interrupt. Tasks can be configured to run on the interrupt thread using the 'interrupts' option. When the start trigger pin supports interrupts, the task is started with a lower delay and jitter: about 35 us delay and 4 us jitter. For interrupts tasks, the ticking of the task is paced by the chip timer with a high timing accuracy (4 us jitter). Note that there is only one interrupt thread: when two interrupt tasks have scheduled action at the same time, the actions will be executed in sequence, so one of them will be too late.

The Arduino UNO has two pins that support interrupts: pin 2 and 3.

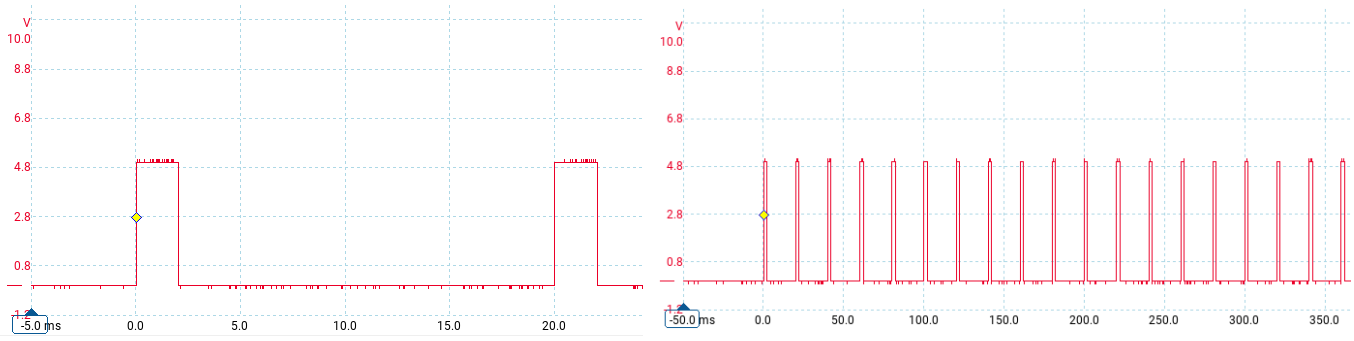
The Arduino Mega has 6 pins that support interrupts: pins 2, 3, 18, 19, 20, and 21.

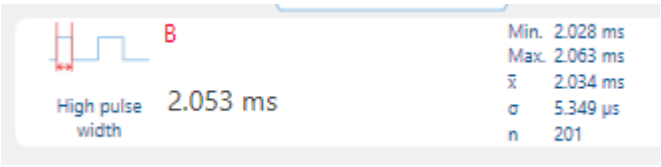
Example 1: Camera Trigger started manually

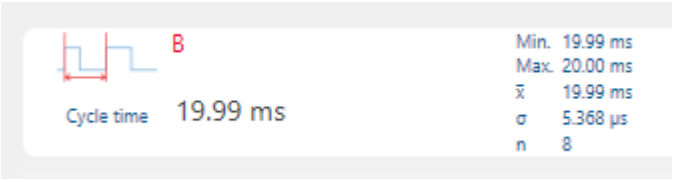
This example shows how to trigger a camera 100 times at 20 ms intervals. Pin ‘BNC 1’ is an output pin. The ‘up’ time is 2 ms, the ‘down’ time is 18 ms, so the time between the triggers is 20 ms.



After defining the task and pressing ‘Send’, select the ‘Control’ page and click on the running icon to start the task. The pulse pattern on BNC 1 shows a jitter of about 5 us:

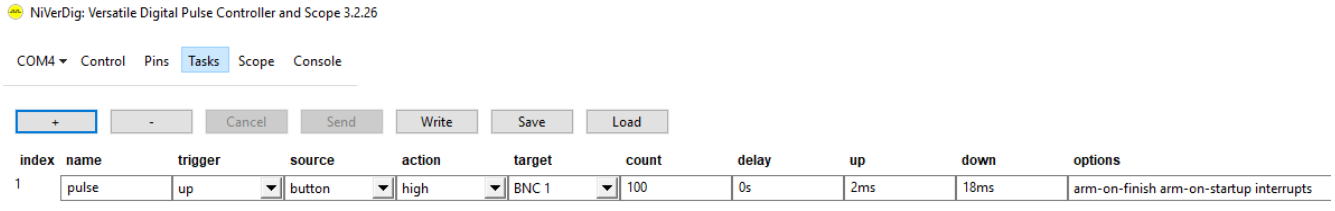






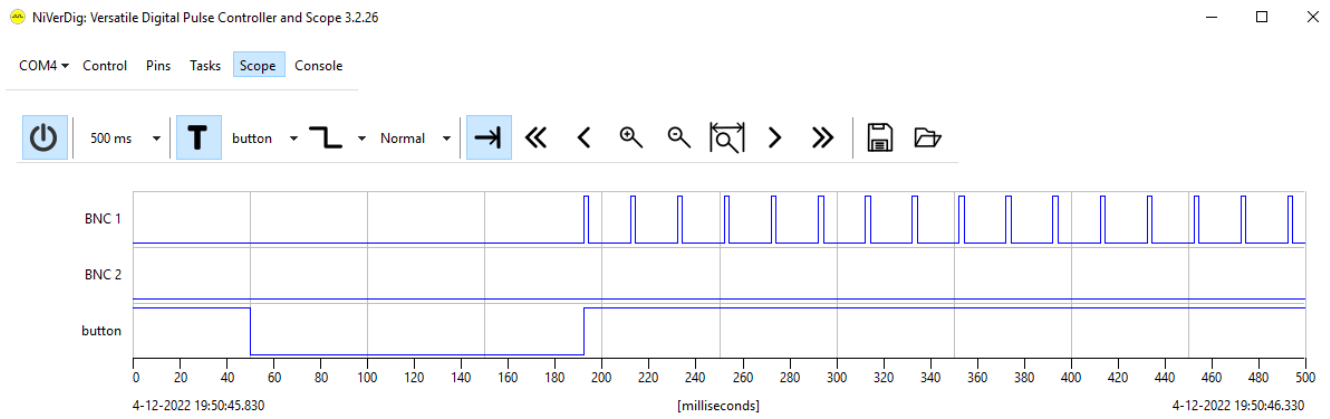
Example 2: Camera Trigger started by button

This example shows how the camera trigger sequence can be started by pressing the button.



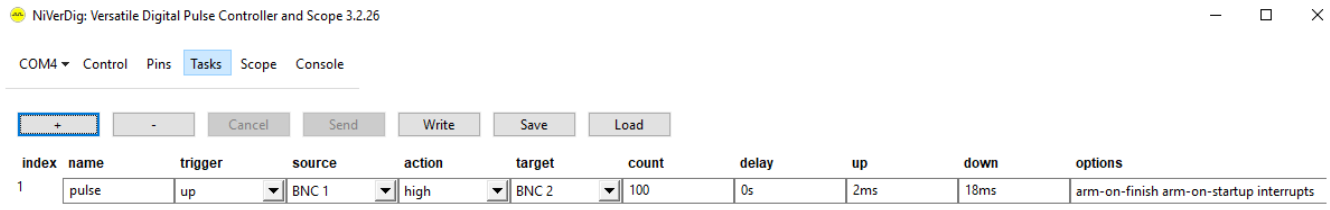
After Sending this task definition to the device, activate the Scope panel, select the 500ms period and press the first button ‘Record’ to start recording. Now press on the button on the device. The graphs

will show that the button line goes low when the button is pressed, followed by the camera trigger pulses on the BNC 1 line:

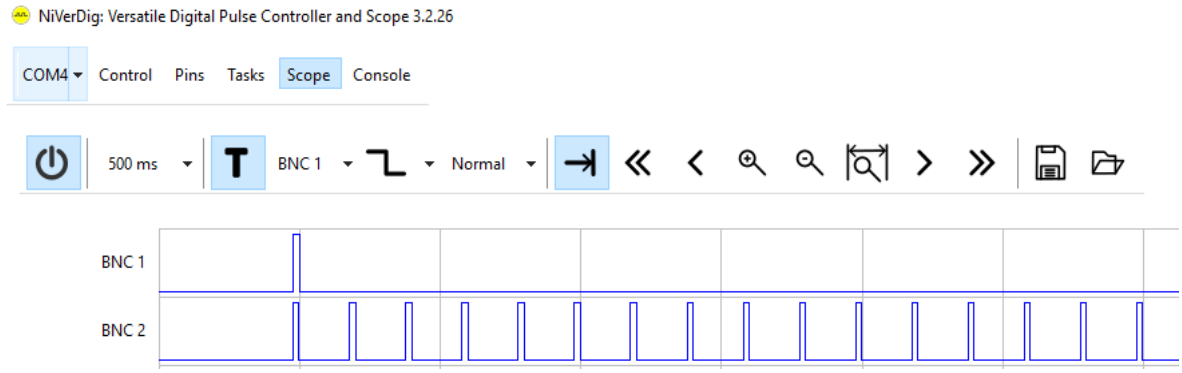


Example 3: Camera Trigger started by external trigger

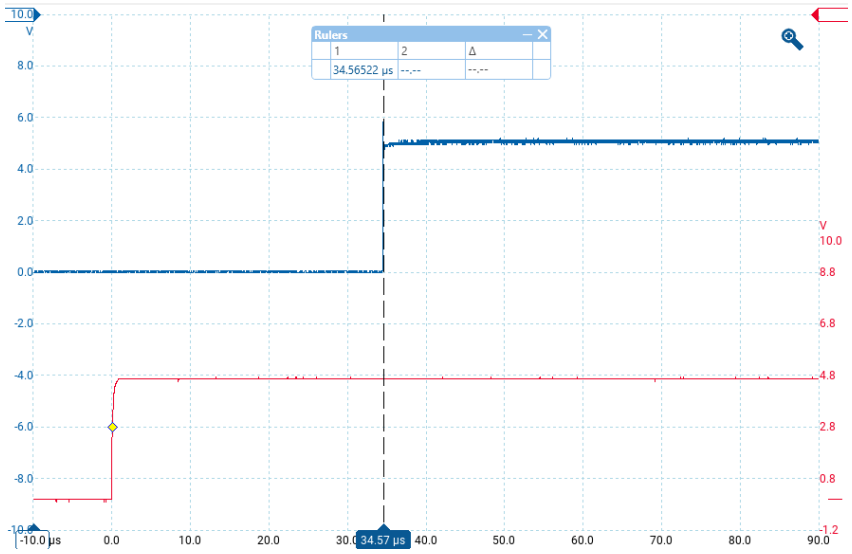
This example shows how the camera trigger sequence can be started by an external trigger.
Pin BNC 1 is configured as input, pin BNC 2 is configured as output.



On the Scope panel, press the T button to enable the trigger and select the ‘BNC 1’ trigger source and the Normal trigger mode. Enable recording with the first button. The camera triggers sequence is generated on the BNC 2 as soon as the trigger on BNC 1 arrives:



The delay of the start by a pulse on a pin that supports interrupts is about 35 us:



Example 4: Stimulation pulse synchronized with camera

This example illustrates how to output a stimulation pulse synchronized with the camera. Without synchronization, a manually started application will occur with a random time difference with the camera frame. To synchronize it with the camera, NiVerDig delays the trigger until the next camera frame detected.

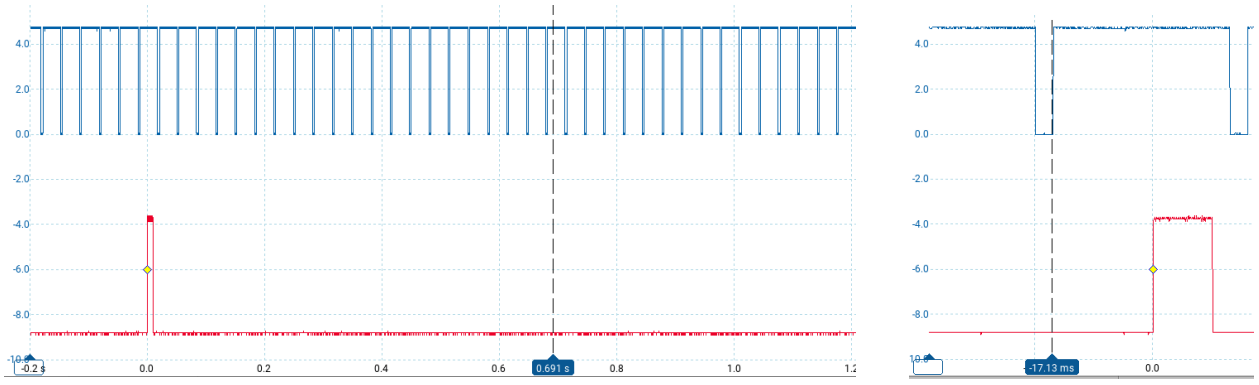
Two tasks are defined: the ‘arm’ task is triggered by pushing the button. This task ‘arms’ the ‘trig’ task. After the ‘trig’ task is armed, it will started by the next camera sync and outputs the stimulation trigger with the specified delay. Note that the ‘arm’ task has the automatic arm options set, but the ‘trig’ task not: it should by ‘idle’ and ignore the camera syncs until ‘armed’ by the button.

NiVerDig: Versatile Digital Pulse Controller and Scope 3.2.26

COM4 ▾ Control Pins Tasks Scope Console

\+ - Cancel Send Write Save Load

index	name	trigger	source	action	target	count	delay	up	down	options
1	arm	up	button	arm	trig	0	0s	0s	0s	arm-on-finish arm-on-startup interrupts
2	trig	up	cam sync	high	stimulate	1	17ms	10ms	10ms	interrupts



Example 5: send camera trigger when wheel is on position

This example illustrates how to send the next trigger to the camera when a filter wheel is on position.
The pin ‘cam trig’ is configured as output and connected to the camera.
The pin ‘whe move’ is configured to the ‘wheel is moving’ signal from the filter wheel.
The task ‘cam trig1’ is started manual and outputs a camera trigger.
The task ‘arm wait1’ is started when ‘cam trig1’ stops and arms the task ‘wait1’.
The task ‘wait1’ waits until the ‘whe move’ goes down and starts the next ‘cam trig1’ run.

The filterwheel must be programmed with other software to move to the next filter position upon the end of the camera exposure.

Start the ‘cam trig1’ task once by software or manually. This will trigger the camera; the filter wheel will start to move to the next position upon the end of exposure; task ‘arm wait1’ will arm the task ‘wait1’; the task ‘wait1’ will generate the next camera trigger when the wheel is on position.

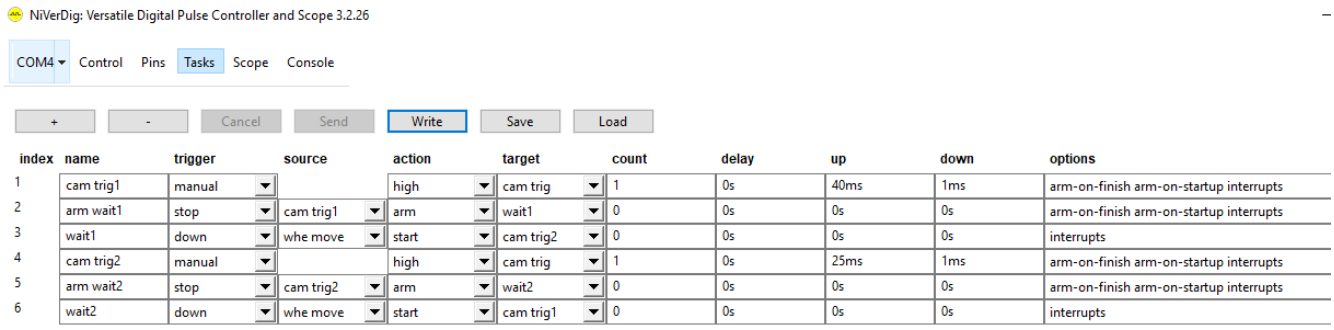
NiVerDig: Versatile Digital Pulse Controller and Scope 3.2.26

COM4 Control Pins Tasks Scope Console

\+ - Cancel Send Write Save Load

index	name	trigger	source	action	target	count	delay	up	down	options
1	cam trig1	manual		high	cam trig	1	0s	20ms	1ms	arm-on-finish arm-on-startup interrupts
2	arm wait1	stop	cam trig1	arm	wait1	0	0s	0s	0s	arm-on-finish arm-on-startup interrupts
3	wait1	down	whe move	start	cam trig1	0	0s	0s	0s	interrupts

This scheme can be extended for more than one phase.
E.G. this is the task definition for two phases exposure. If the camera is configured in ‘bulb’ mode, the trigger signal determines the exposure time. The ‘up’ time for the first camera trigger task is different than for the second to implement two different exposures per channel.



index	name	trigger	source	action	target	count	delay	up	down	options
1	cam trig1	manual		high	cam trig	1	0s	40ms	1ms	arm-on-finish arm-on-startup interrupts
2	arm wait1	stop	cam trig1	arm	wait1	0	0s	0s	0s	arm-on-finish arm-on-startup interrupts
3	wait1	down	whe move	start	cam trig2	0	0s	0s	0s	interrupts
4	cam trig2	manual		high	cam trig	1	0s	25ms	1ms	arm-on-finish arm-on-startup interrupts
5	arm wait2	stop	cam trig2	arm	wait2	0	0s	0s	0s	arm-on-finish arm-on-startup interrupts
6	wait2	down	whe move	start	cam trig1	0	0s	0s	0s	interrupts

Example 7: Record timestamps of events

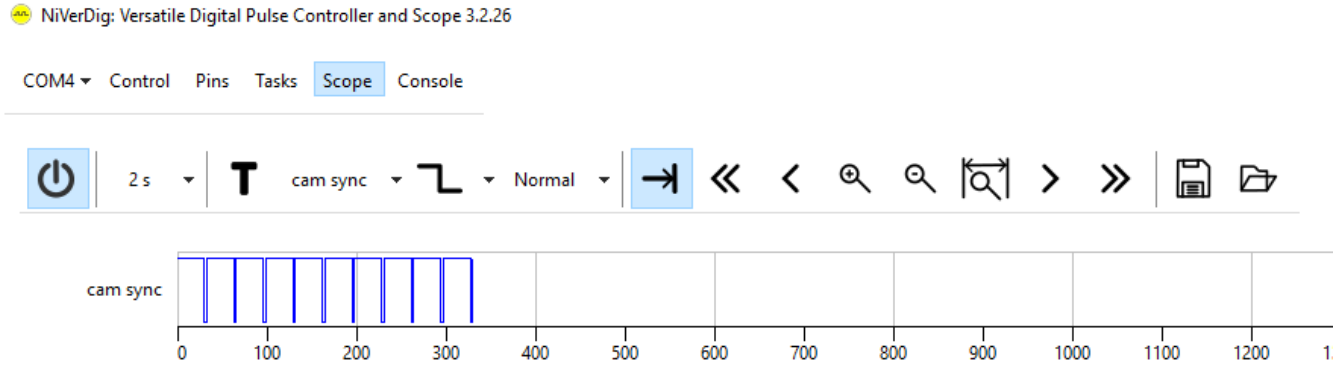
This example illustrates how to record the timestamps of TTL events.
Configure the pins as input and delete all tasks.
Select the Scope panel and press the most left button ‘Record’.

Now all events detected on the input pins will be recorded to a temporary file.

To save the events upon end of the recording, press the Save button (diskette icon) and specify a format and name.

The following formats are available:

name	details
NiVerDig Binary Event Format nkbef	binary: 8 bytes filetime, 1 byte channel, 1 byte state
NiVerDig Text Event Format nktef	text: log time format, channel and state
NiVerDig Relative Timing Format nkref	text: relative time in microseconds, channel and state.



The text formats start with a list of pin indices and names, followed by the events with for every event a line with the time, pin-index and state:

```
pin     0       cam sync
time    pin     state
2022-12-04 21:13:54.105000      0       1
2022-12-04 21:13:54.111028      0       0
```

Command-Line interface

The Arduino is accessed through a COM port. The 'Console' panel shows the text sent to and received from the device. Control of the device is also possible from other programs.

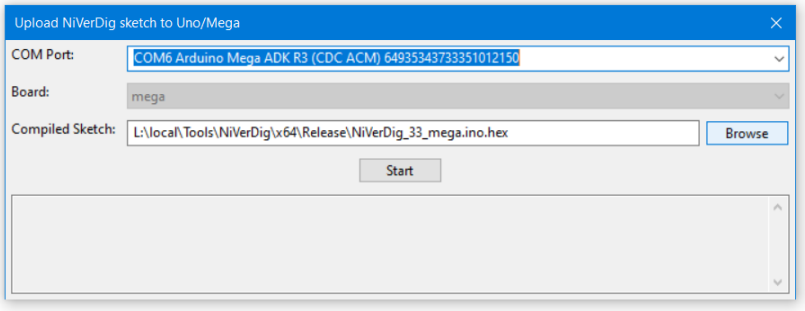
Connection details: baud-rate 500000 bps, 8-bits, 1 stop bit, parity: none, flow-control: none, end-of-line character: newline

command	details
?	lists all available commands
halt [n]	stops (0) or resumes (1) task execution. show current halt status without argument
start n	starts task n
stop [n]	stops task n (all tasks without argument)
arm n	arms task n
disarm n	disarms task n
pin [n [?]]	reports the state of pin n or all pins
pin [n [s]]	sets pin n to state s s not specified: show the current state of pin n n not specified: show the state of all pins
dpin ?	shows the pin definitions
dpin -[*]	decreases the number of defined logical pins the * argument deletes all poins
dpin n	configures logical pin n: n must be the index of a defined pin or one higher there are argument syntaxes: one to define all properties and one to define only one: dpin <index> <name> <pin> <mode> <init> <toggle>: define pin dpin <index>|<name> <setting> [=] <value>: change one property <index> : [1 to N] <name> : quoted pin name [9] <pin> : [0 to N] <mode> : (output|input|pullup|pwm|adc) <init> : <output> [0 to 1] (low|high) <pwm> [0 to 255] <toggle> : <pwm> [0 to 255] Note: if the second argument is one of the property names, the second syntax is assumed. Don't define a pin name that is equal to a property name.
task [n [s]]	sets task n to state s (0: idle, 1: armed, 3: running) note: auto-arm and auto triggered task will start automatically after they are stopped s not specified: show the state of task n n not specified: show the state of all tasks
dtask ?	show the task definitions
dtask -[*]	decreases the number of defined tasks the * argument deletes all tasks
dtask n	configures task n: n must be the index of a defined task or one higher there are argument syntaxes: one to define all properties and one to define only one: task <index> <name> <trigger> <source> <action>

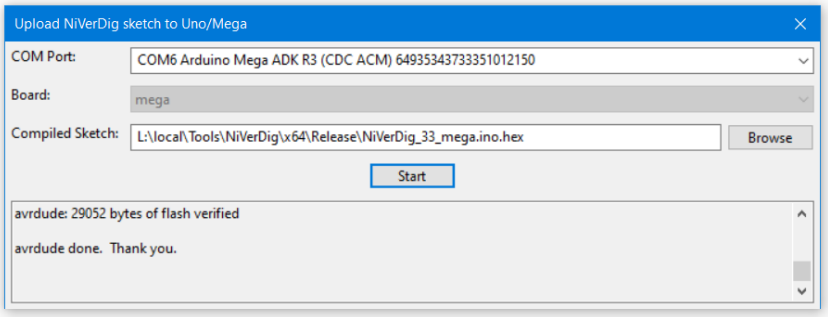
<target> <count> <delay> <up> <down> <options>
dtask <index>|<name> <property> [=] <value>
The property definitions are:
<index> : [1 to N]
<name> : quoted task name [9]
<trigger> : <software> (auto|manual) <input-pin> (up|down|any|high|low)
<in-task> (start|stop)
<source> : <input-pin> (input pin-index or pin-name)
<in-task> (task-index or task-name)
<action> : <output-pin> (low|high|toggle)
<out-task> (arm|start|restart|stop|kick) <none> (none)
<target> : <output-pin> (output pin-index or pin-name)
<out-task> (task-index or task-name) <none> ()
<count> : [-1 to 1073741820] repeat count: -1 for continuous, 0 for single action
<delay> : n[s|ms|us] delay: 0 or between 100 us and 17:53
<up> : n[s|ms|us] up time: 0 or between 100 us and 17:53
<down> : n[s|ms|us] down time: 0 or between 100 us and 17:53
<options> : (arm-on-finish arm-on-startup interrupts)
Note: if the second argument is one of the property names, the second syntax is assumed. Don't define a task name that is equal to a property name.

Sketch Upload

The precompiled Arduino NiVerDig Sketch can be uploaded to a Uno or Mega board using AVRdude. From the Ports dropdown menu select 'Upload NiVerDig Sketch to Uno/Mega'. Select the appropriate COM port, the board model and the matching sketch. Press Start to start the upload.



On completion, the software will connect to the board:



NIS Macro

NiVerDig.mac

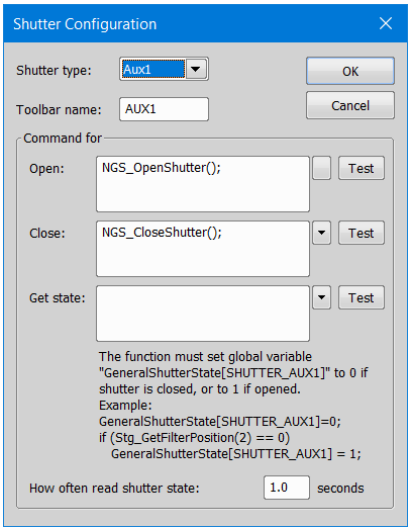
Save this code in c:\program files\nis-elements\macros\NiVerDig.mac

```
global long NVD_port;

int main() { }

int NVD_OpenPort(int port)
{
	NVD_port = port;

	// NIS does not support a baudrate of 500000, use the nearest supported rate
	OpenPort(NVD_port, 460800, 8, "N", 1);

	// Arduino needs 1 second to boot
	Wait(1.0);
	// read away the 'hello' message
	NVD_ReadAllLines();
}

int NVD_ClosePort()
{
	NVD_ClosePort(NVD_port);
	NVD_port = 0;
}

int NVD_SetPin(int pin, int state)
{
	char command[64];
	sprintf(command, "pin %d %d", "pin,state");
	WritePort(NVD_port, command, 1, 0);
}

int NVD_GetPin(int pin)
{
	char command[64];
	char8 answerA[64];
	int pos;
	pos = -1;
	sprintf(command, "pin %d ?", "pin");
	WritePort(NVD_port, command, 1, 0);
	ReadPort(NVD_port, answerA, 64);	// read the answer
	if (answerA[0] == '1') pos = 1;
	if (answerA[0] == '0') pos = 0;
	return pos;
}

int NVD_SetTask(int task, int state)
{
	char command[64];
	sprintf(command, "task %d %d", "task,state");
	WritePort(NVD_port, command, 1, 0);
}

int NVD_ReadAllLines()
{
	char8 answerA[64];
	while (ReadPort(NVD_port, answerA, 64) > 0)
	{
	}
}
```

NiVerDigGeneralShutter.mac

Save this code in c:\program files\nis-elements\macros\NiVerDig-GeneralShutter.mac and configure it to be executed on NIS startup (Macros | Run Macro on Event).

Add a 'General Shutter' device in the device manager with the configuration as shown on the right.




```
// AUX1 general shutter on COM4 NiVerDig pin 3
int main()
{

   if (!ExistProc("NVD_OpenPort"))
   {
      RunMacro("c:/Program Files/NIS-Elements/macros/NiVerDig.mac");
   }

   NVD_OpenPort(4);

   // NGS_OnTimer();
   Timer(100, 5000, "NGS_OnTimer()");
}

int NGS_OpenShutter()
{
   NVD_SetPin(3, 1);
}

int NGS_CloseShutter()
{
   NVD_SetPin(3, 0);
}

// NIS 5.42.02 bug: General Shutter GetState is never called: just use a Timer !
int NGS_OnTimer()
{
   int pos;
   pos = NVD_GetPin(3);
   if (pos == -1) return 0;
   if (pos == GeneralShutterState[SHUTTER_AUX1]) return 0;
   Stg_SetShutterStateEx("AUX1", pos);
}
```

Build your own Hardware

To build your own NiVerDig, order a (compatible) Uno or Mega board, connect the required interface parts and upload the NiVerDig precompiled sketch.

The Mega has more dynamic memory, allowing more pin and task definitions.

The capabilities of the Uno and Mega:

	Uno	Mega
Frequency	16 MHz	16 MHz
pins supporting interrupts	2: pins 2,3	6: pins 2,3,18,19,20,21
dynamic memory	2048 bytes	8192 bytes
NiVerDig pin definitions	4	8
NiVerDig task definitions	52	52

The NiVerDig sketch allows to define the pins freely, but on a factory reset (button pressed 5 seconds on boot), the default pin definitions are restored.

To limit the current on shortcut of the TTL ports, 1 kΩ resister can be used. This will have no significant effect on the speed performance. The LED is connected with resisters that limit the emission to non-disturbing weak level. Determine the best values by trial and error.

These are the default configurations:

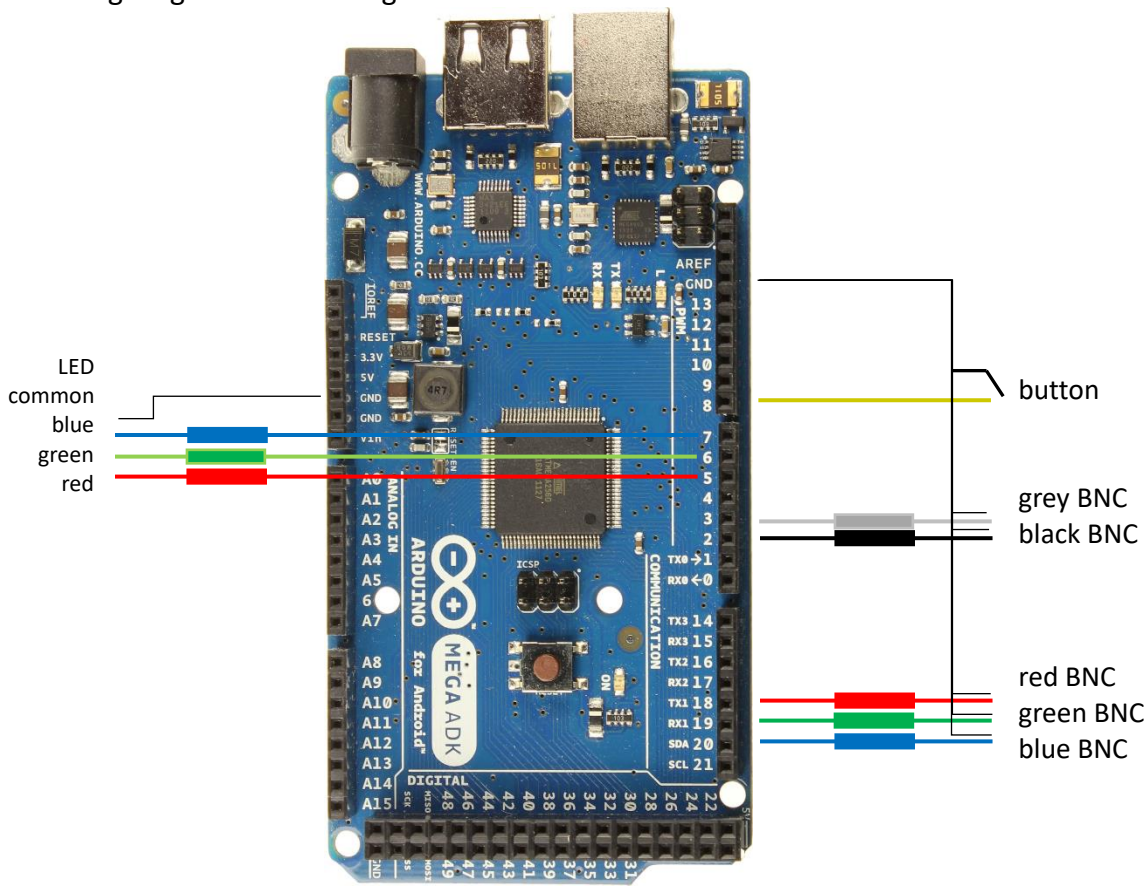
Uno

Pin	Resistor	Mode	Name
2	1 kΩ	INPUT	BNC in
3	1 kΩ	OUTPUT	BNC out
8	-	PULLUP	button

Mega

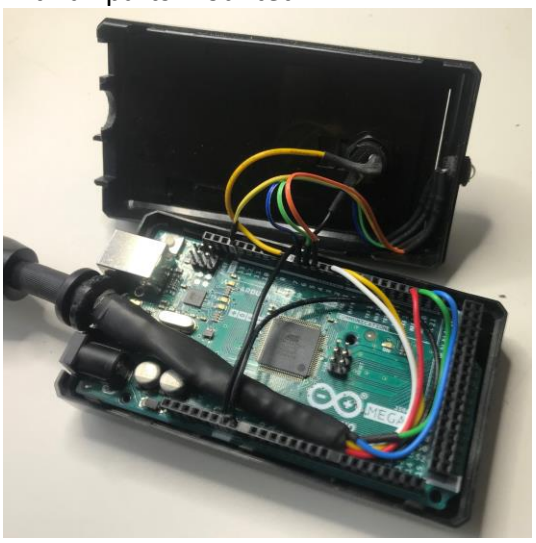
Pin	Resistor	Mode	Name
2	1 kΩ	INPUT	black BNC
3	1 kΩ	INPUT	grey BNC
18	1 kΩ	INPUT	red BNC
19	1 kΩ	INPUT	green BNC
20	1 kΩ	INPUT	blue BNC
5	1 kΩ	OUTPUT	red LED
6	10 kΩ	OUTPUT	green LED
7	2.2 kΩ	OUTPUT	blue LED
8	-	PULLUP	button

The wiring diagram for the Mega:

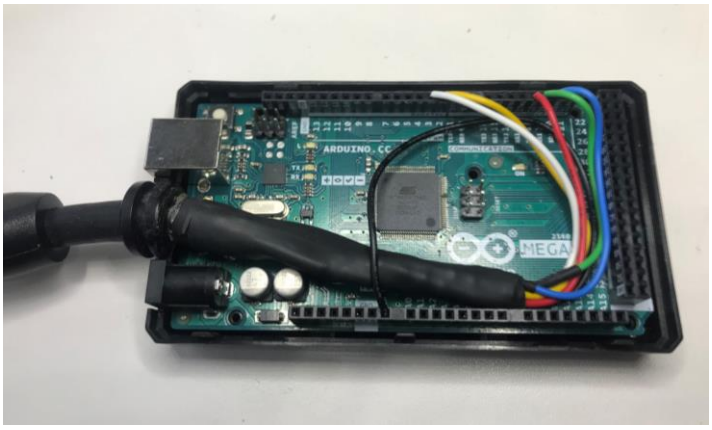


Pictures of the assembly process (in reverse order):

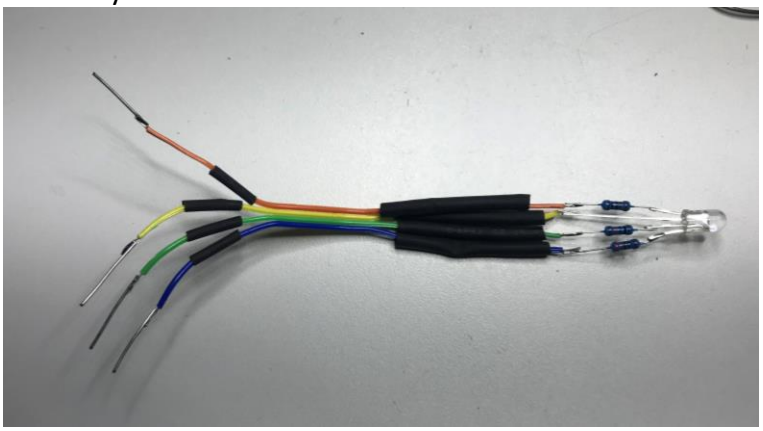
with all parts mounted:



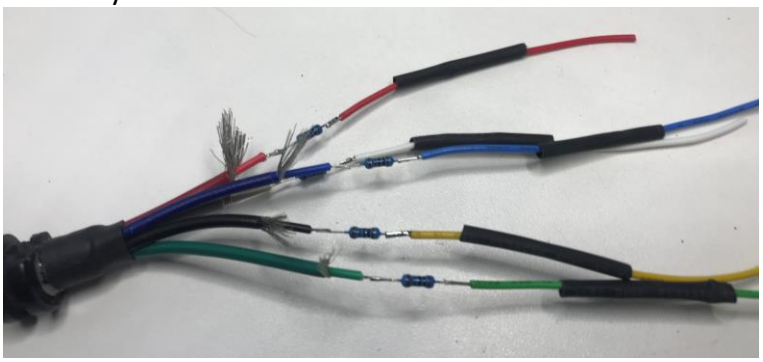
with only the cable mounted:

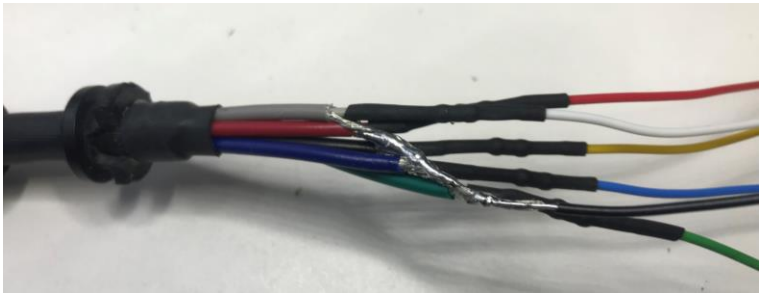


assembly of the LED:



assembly of the cable:





Colofon

Author: Kees van der Oord <Kees.van.der.Oord@inter.nl.net>

Date: Jan 15, 2023
Version: 33