Using hardware Gate and Delay of the V2495 / DT5495 in SciCompiler project

Abstract

This application note will demonstrate how to use the hardware Gate and Delay of the X2495 board to shape a digital signal introducing a programmable delay and width.

The feature exploit the V2495/DT5495 hardware gate and delay generator to implement a jitter-free delay. The demonstration requires a X2495 board, a signal generator and an oscilloscope.

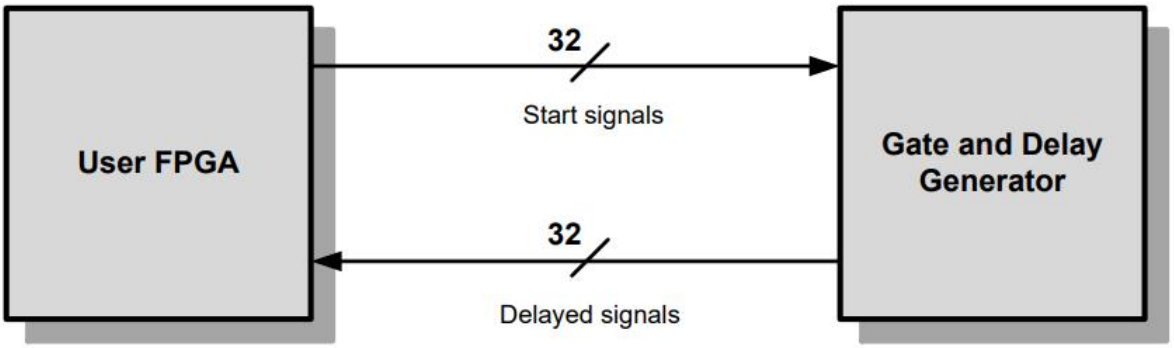
Due to the usage of a particular hardware component of the X2495 this code can not work on different hardware solution

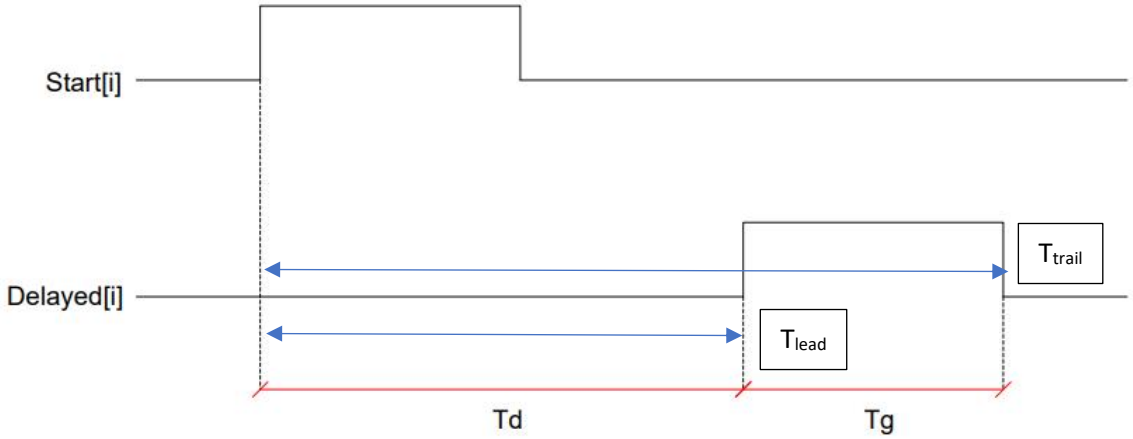
Files related to this application note can be downloaded from

https://github.com/NuclearInstruments/SCIAN1001-V2495-GD

1 Hardware Gate and Delay

The V2495 hosts a Gate and Delay Generator able to provide up to 32 gated and delayed signals ("delayed signals") triggered by 32 inputs ("start signals"). The gate width and delay value are user programmable. The GDG is an external component implemented in a Xilinx Spartan-6 FPGA. It is connected through a serial bus (SPI) to the User FPGA for gate and delay register programming.





The signals are sent to the GDG, where they are gated and delayed according to the user settings and then returned to the UFPGA. The width of the delayed signal only depends on the gate value set and is not related to the width of the incoming signal. The gate and delay values are 16-bit wide and their sum cannot exceed 65535 (0xFFFF). They are both generated by using an internal clock, which computes the number of clock cycles between the incoming signal and the leading and trailing edge of the gate signal. The times of occurrence of the leading (T_{lead}) and trailing (T_{trail}) can be obtained using the following formulas, where N_g(N_d) is the gate (delay) value set by register and T_0, T_1 are the time values reported in table

PARAMETER	VALUE
T_0	12 ns (typ.) ± 10%
T_1	10.7 ns (typ.) ± 10%
T_{max}	$T_0 + T_1 (2^{16}-1)$

T_{lead} = 0 if $N_d=0$

$T_0+T_1(Nd-1)$ if $N_d>0$

T_{trail} = 0 if $N_d=N_g=0$

$T_0+T_1(N_d+N_g-1)$ if $(N_d+N_g)\geq 1$

$T_{delay}=T_{lead}$

$T_{gate}=T_{trail}-T_{lead}$

T_1 represents the minimum increment of either the gate or the delay value. The presence of T_0 is due to a slightly longer duration of the minimum delay time increment (or gate increment, if the Ng=0) with respect to the standard T_1 increment. It should be noted that both T1 and T0 can vary from channel to channel, with an expected variation interval of ±10% for both. Please also consider that, when setting Nd=0, the observed delay of the outgoing signal with respect to the incoming one is due to the paths on the PCB and inside the FPGAs, which are delay channel-dependent.

2 Preliminary setup

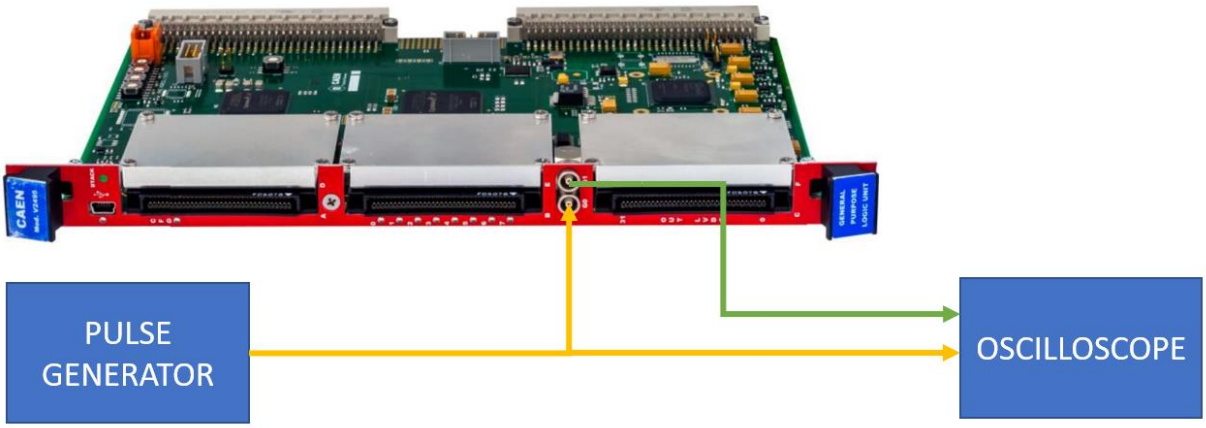
First make sure to have all the relevant connections in place and the latest available software upgrades installed.

Connect a pulse generator to the Port G-0 of the V2495 board. Split the output of the pulse generator with a T splitter and connect one output to the V2495 G0 the other output to channel 1 of an oscilloscope.

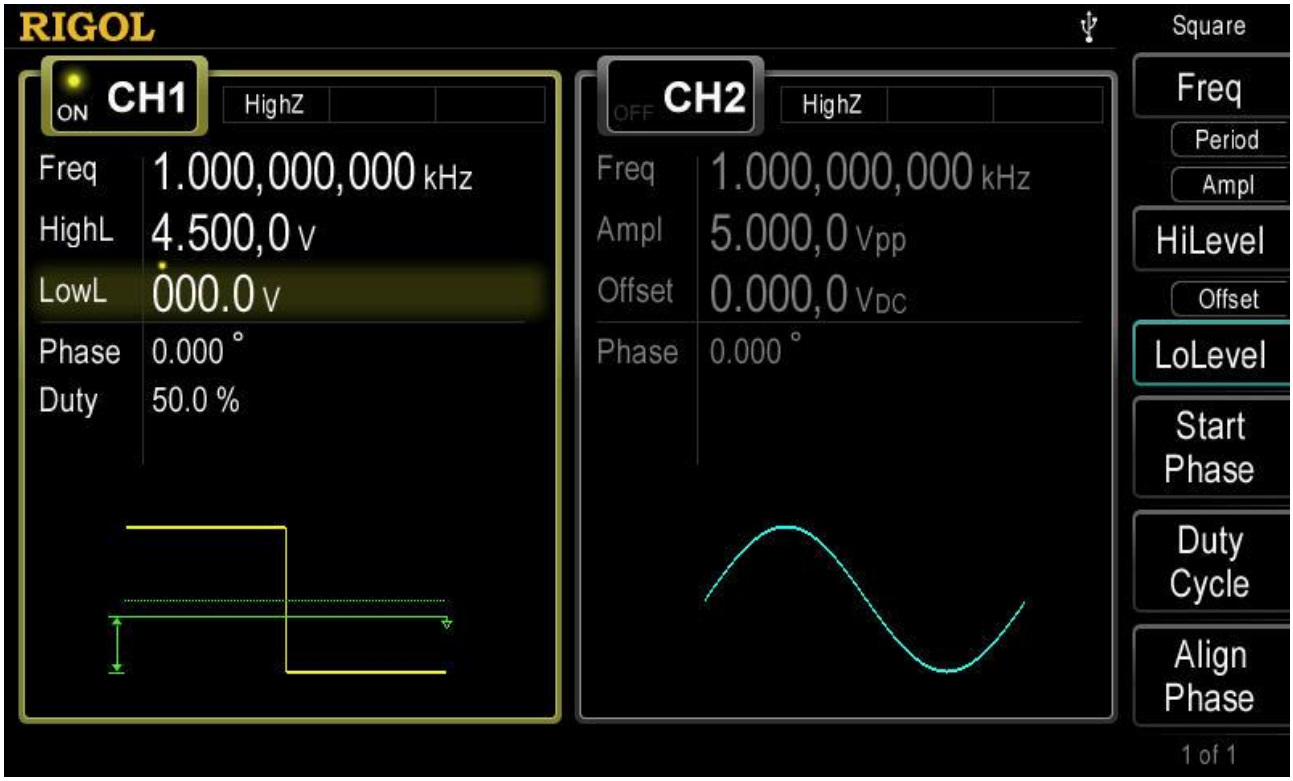
Connect port G1 of the V2495 (output) to the input channel 2 of the oscilloscope.

In this way we will see on the two channel of the oscilloscope the signal generated by the pulse generator and the signal generated by the V2495 board after Gate and Delay

This is a reference of how your work area may look like:

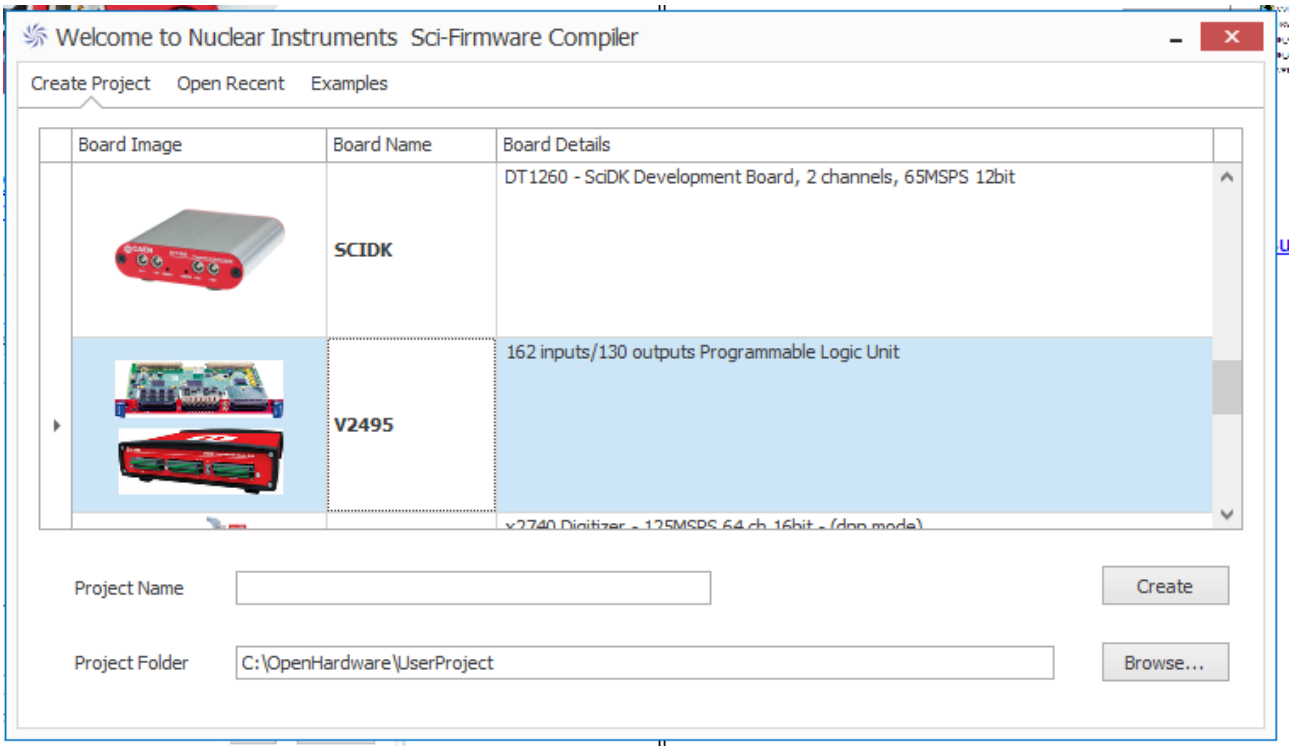


Se the signal generator to generate a square wave 1KHz, TTL, 50% duty cycle



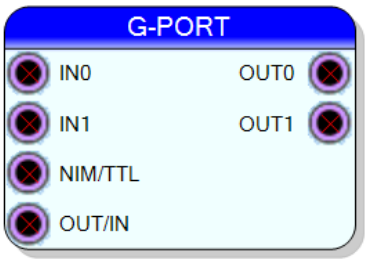
SciCompiler project setup

Now open SciCompiler, select *New Project*, insert the name of the design we're going to create, select V2495 as board

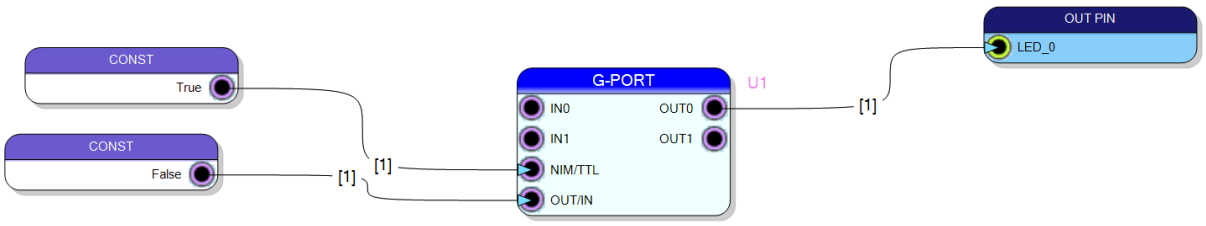


3 Gate and Delay implementation

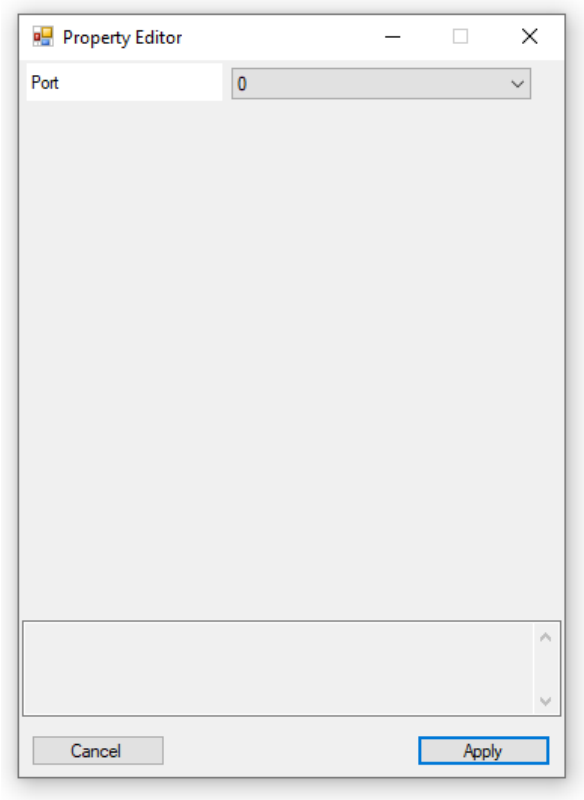
Under the tool box menu (or right click in the project area) peek from board pin the G-PORT block. This will instantiate the G port in the design. The G-PORT as 2 lemos and we want to use one as input and the other as output



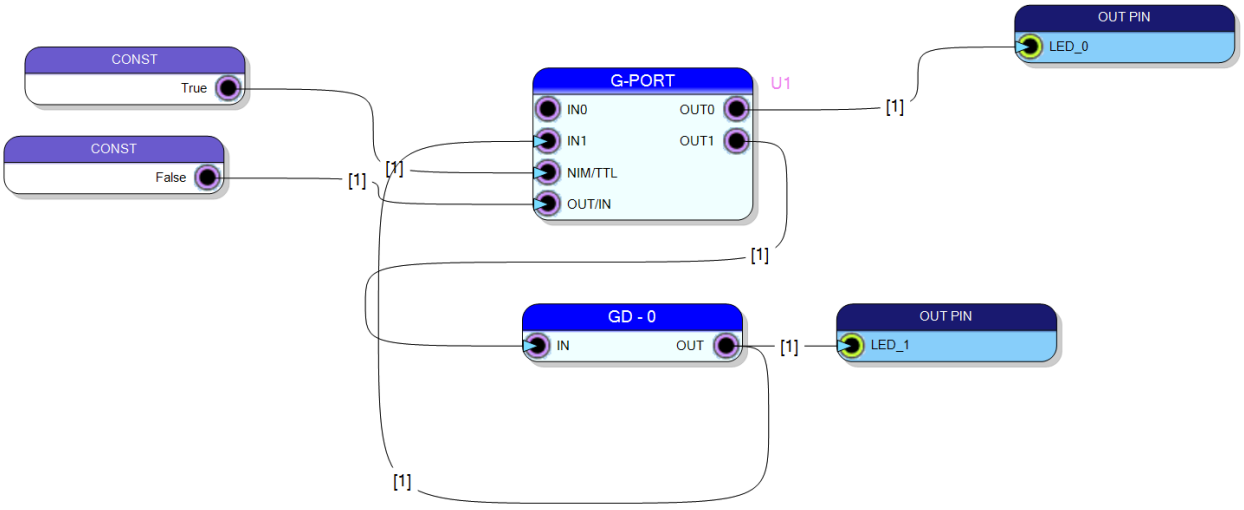
In order to use one G-PORT as input and the other as OUTPUT, set the OUT/IN pin as OUTPUT (Low) and force LOW the INx pin of the G-PORT so we configure it as following



Then we peek from board pin the Gate and Delay block. The tool ask as to specify which of the channel of the G&D use. Pay attention to do not use multiple time the same channel. There are 32 channels and user must care to use each one for no more than one IN/OUT pair



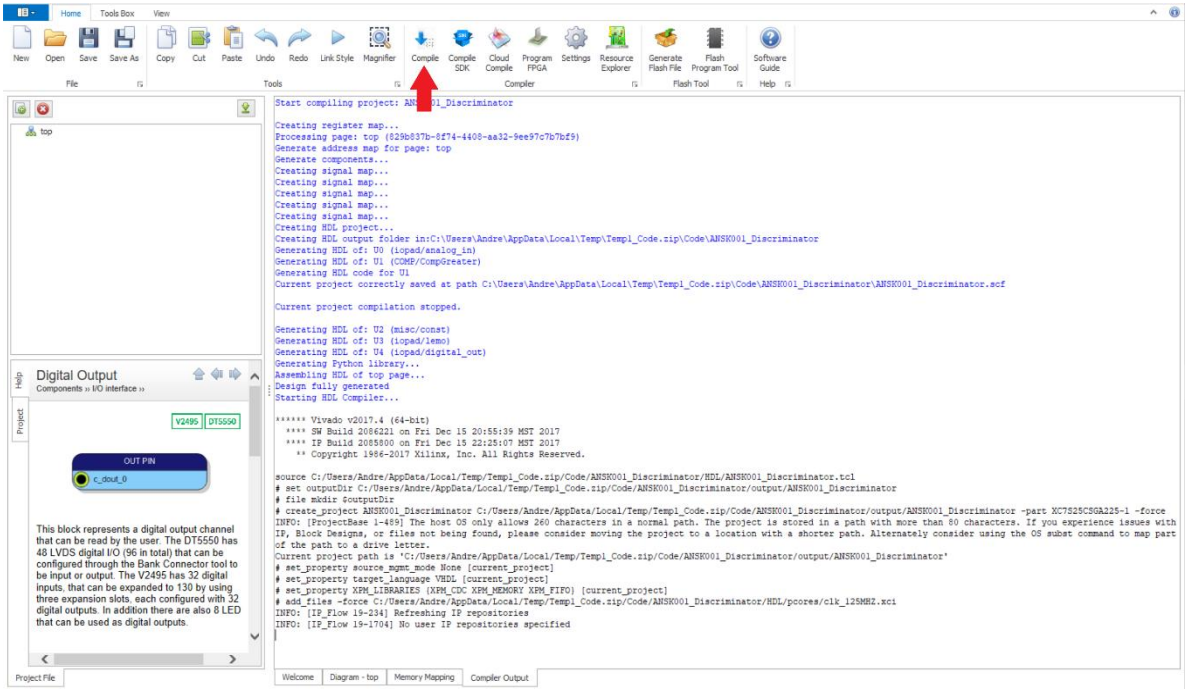
Connect the G&D input to the G0 out and G&D output to the G1 IN



4 Compilator

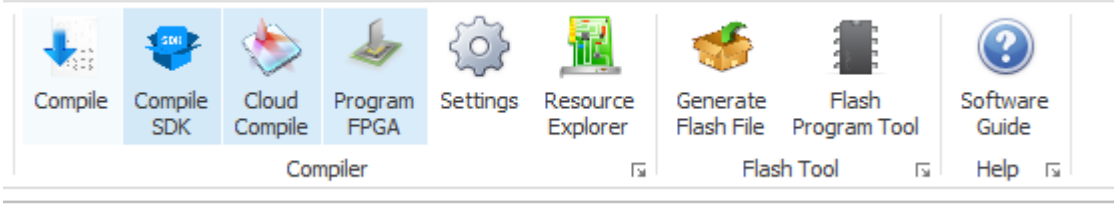
Once the design is done you can leave the *Tools Box* for the *Home* menu bar, where you will click *Compile*.

SciCompiler will prompt a confirmation box and then starts to sort out your design into an Hardware Description Language and further proceed to invoke Vivado generating the full bitstream needed to run your design on the FPGA:



As soon as you get the message `Successful compilation!` at the end of the bitstream generation the design has been fully finalized.

Press the button Generate Flash File to generate the firmware to be downloaded on the board the can upload tool



5 Board configuration

Use the CAEN firmware upgrade tool to download the firmware on the board. Refer to the V2495 user manual UM5175 Section 8.2 CAEN Upgrader

6 Write the C++ code to control the Gate and Delay

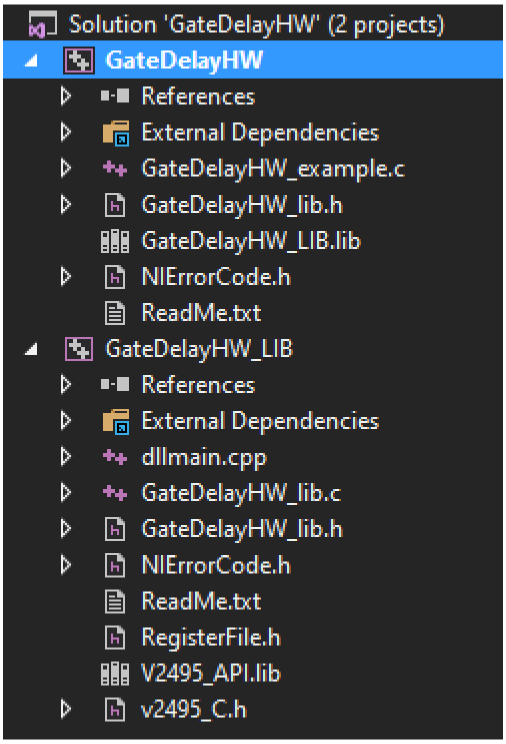
The SciCompiler generate in the project folder a library folder that contains files to interface via USB the board with a PC

Open visual studio project in library/C/lib/VC++/GateDelayHW.sln

The solution is composed by two project:

- Library
- Example code

While the library in ready to use the example code contains just the connection to the device.



Compile the library by right it on GateDelayHW_LIB and select build

You should get a successful massage like this

```
1>------ Build started: Project: GateDelayHW_LIB, Configuration: Debug Win32 -
1>C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\V140\Microsoft.CppBuild.ta
1>  GateDelayHW_lib.c
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>c:\openhardware\userproject\gatedelayhw\library\c\lib\src\gatedelayhw_lib.c(
1>  dllmain.cpp
1>dllmain.obj : warning LNK4075: ignoring '/EDITANDCONTINUE' due to '/OPT:LBR'
1>     Creating library bin\GateDelayHW_LIB.lib and object bin\GateDelayHW_LIB
1>  GateDelayHW_LIB.vcxproj -> c:\OpenHardware\UserProject\GateDelayHW\library
1>  GateDelayHW_LIB.vcxproj -> bin\GateDelayHW_LIB.pdb (Full PDB)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Replace the Example code with the following code.

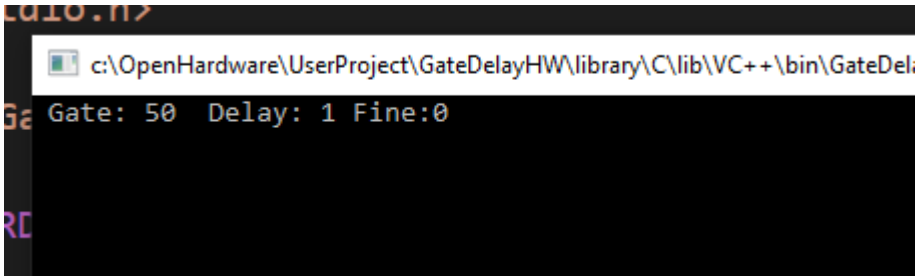
Please pay attention to write the correct board serial number in the `BOARD_SERIAL_NUMBER` define

```
#define BOARD_SERIAL_NUMBER "0020"
```

The code will be like this:

```
#include "V2495_C.h"
#include  <stdlib.h>
#include <stdbool.h>
#include <stdio.h>

#include  "GateDelayHW_lib.h"

#define BOARD_SERIAL_NUMBER "0020"



int main(int argc, char* argv[])
{
	int	handle;
	int ret;
	NI_RESULT res;
	UINT32	val;
	UINT32 gate;
	UINT32 delay;
	UINT32 fine_tune;
	V2495_Startup();
	if (V2495_AttachNewDevice(0, BOARD_SERIAL_NUMBER, 0, &handle) != NI_OK) {
printf("Unable to connect to the board!\n"); return (-1); };

	GateDelay_Calibrate(&handle);
	GateDelay_Bypass(0, 0, &handle);		//Disable delay bypass
	GateDelay_Program(0,
		50,					//GATE
		1,					//DELAY
		0,					//FINE
		1,					//ENABLE
		&handle);
	GateDelay_Read(0,
		&gate,
		&delay,
		&fine_tune,
		&handle);
	printf(" Gate: %d  Delay: %d Fine:%d\n", gate, delay, fine_tune);

	while (1);

	return 0;
}
```

Run the application and a prompt windows will show you the parameter programmed in the delay, fine and fine tune.



Look at the signal on the oscilloscope and appreciate the shape of the output of the the V2495 changing if function of the DELAY, GATE and FINE_TUNING parameter set by the `GateDelay_Program` function

Modify the code introducing a loop on the FINE_TUNE to appreciate the fine variation of the delay/gate modifying the fine parameter

```
while (1) {

	for (int i = 0; i < 255; i++) {
		GateDelay_Program(0,
			50,			//GATE
			1,			//DELAY
			i,			//FINE
			1,			//ENABLE
			&handle);
	}


}
```

7 Write the Python code to control the Gate and Delay

You can repeat the same exercise done in c++ in python.

Replace the example code with the following code.

```
from GateDelayHW_Functions import *
from ctypes import *


[ListOfDevices, count] = ListUSBDevices()
if (count > 0):

    board = ListOfDevices[0].encode('utf-8')

    Init()
    EthConnection = 0
    TCPPort = 0
    UDPPort = 0
    [err, handle] = ConnectDevice(EthConnection, board, TCPPort, UDPPort)
    if (err == 0):
        print("Successful connection to board ", board)
    else:
        print("Connection Error")

    GateDelay_Calibrate(handle)
    GateDelay_Bypass(0, 0, handle)  #Disable delay bypass
    GateDelay_Program(0,
                50,                 #GATE
                1,                  #DELAY
                0,                  #FINE
                1,                  #ENABLE
                handle)

    err, gate, delay, fine_tune    =  GateDelay_Read(0, handle)
    print (" Gate: " + str(gate) + " Delay: " + str(delay) + " Fine: " +
str(fine_tune) + "\n")
```

This application note ends here