CompLex
(VC-SPG)

An OpenSoundControl (OSC) and
Voltage Controlled Signal Path Generator

Lex van den Broek

Master Thesis
Instrument & Interfaces
Institute of Sonology - Steim
Royal Conservatoire, The Hague (NL)
May 2017

Supervisor: Edwin van der Heide

Abstract

This thesis describes my electronic and artistic research into the design of what I have called the Voltage Controlled Signal Path Generator (VC-SPG). It is a switching audio matrix that can both be seen as a new tool, and as a part of a musical-instrument. As we will see, it can be applied in an analogue electronic music studio setup, a modular synthesizer or in an interactive art-installation. This master research project is a continuation of my previous work at the Royal Conservatoire where I design new technology for Art and Education.

The fundamental core of my research project is the development of an audio-matrix with 16 inputs and 16 outputs that can be fully configured, controlled and programmed with Open Sound Control (OSC) and that can be synced and triggered with external analogue signals. In its present state the VC-SPG has become a new type of generator that is able to switch between different studio presets and form the core of new audio experiences and new compositional approaches.

I will describe and reflect upon both the technical challenges and development and the musical and artistic results shared with me by students and professionals who used the VC-SPG over the last year for their own work. They all experienced the VC-SPG to be adding a new dimension to their creative process. We can conclude that the VC-SPG is not only a new practical tool, but also a creative instrument for electronic-music and art.

Acknowledgements

A project like this can't be developed successfully without the help and input of dedicated coaches, colleagues, students, family and friends. Thank you all for helping me. I want to thank some people who really spend time and effort to give my thoughts new direction.

Lots of thanks to my wife Anita - she is a great support. Edwin, thanks for being my supervisor, for coaching and support and for pushing my thoughts into new directions - I needed that! Johan and Robert, I liked our fruitful brainstorm sessions. Let's continue these sessions! My coaches at Steim, Kristina and Joel. Thank you for the inspiring chats.

Since I'm always in my office at the Conservatoire developing, programming, measuring, testing, teaching and coaching, a lot of people drop by on weekly bases for a coffee, a favor or a chat. Many of these chats were of great help for my research as well. Sometimes it's about the things that are not being said, that triggers my thoughts and ideas. My colleagues at the EWP, thanks for your support and giving me time to do this project.

Kees, Marko, Paul, Siamak, Richard, Frank, Jo, Kasper, Justin, Raviv, Peter, Ruben, Guiliano, Andrea, Kyriakos, Chris, Rolf, Diana, Karst, Kacper and Max:
Thanks!

Content

Content (continued)

1 Introduction

Working in an analogue electronic music studio environment requires working with patch cables enabling the studio user to interconnect all types of sound modules in order to generate sounds. Modular synthesizer setups also depend on making inter-connections first, before all the aspects of the sounds can be explored. To this day the connections or patches in electronic studios or modular-synths are created with wires connecting inputs and outputs. These connections are still static.

In this master-research project I will introduce and explore the electronic design of a new type of instrument, that is capable of changing these hardwired connections, and can dynamically switch between presets controlled by a remote computer or by external signals.

What would the influence be in the way electronic music is composed when this signal path generator will be part of an electronic-studio? Will the introduction of this generator give the composer a new tool to compose electronic music and if so, how will it be applied? To be able to answer these questions, I have built multiple models of the VC-SPG (Voltage Controlled Signal Path Generator) to give users the opportunity to test, evaluate and explore its potential.

1.1 Analogue studio configuration

As head of the Electronics Workshop (EWP) at the Royal Conservatoire, I experience a lot of studio-configurations in general, but especially electronic studios. Electronic studios, much like modular synths, consist of a lot of different sound devices, effect processors and individual modules and in combination with sound-recorders, computers, audio-mixers and loudspeakers new sounds and new music can be explored and created.

The design of the analogue studio of Sonology, which is a classic voltage controlled studio and is based upon the use of an analogue patch panel. See figure 1.1. Working in this studio means you have to make the right interconnections between different modules and make setting and adjustments to create effects like modulation, pitch change, timbres, volume-change and many more effects and sounds.

Figure 1.1. Analogue studio Bea-5 with the patch-panel (on the right) where all ins and outs are located.

In the first year of the master research project, Kees Tazelaar[1] introduced me to the

fundamental control-voltage techniques and explained to me the analogue-studio philosophy. Since the original project title was to design a patch-generator, the difference between the word 'patch' and hardwired connections are a source for discussion. According to Kees, and I agree with him on this point, a patch is not only a hardwired connection between two points, but it is a combination of the connection with the right settings of the linked modules. If the frequency range of a generator and the wave-shape selection is changed, also the resulting sound will change and that change of sound is independent of the hardwired connection. In other words, not only the connection or routing of the wires make the sound, it's the combination of the settings and the connections together that define the word 'patch'.

The patch-cables provide hard connections between inputs and outputs of the different sound- and effect-modules (e.a. triggers or control-voltages). The output of one module can drive the input of other module(s). The patch itself, consisting of multiple cables and the right settings, determines which inputs and outputs are inter-connected and defines the range of sounds that are created in the studio-setup. Currently the position of the patch-cables can only be changed manually. The patch itself is static and will only change if the user changes the connections manually.

My research involves the design and realization of an interface that can switch in total 256 connections or 16 inputs and 16 outputs simultaneously. It can switch the connections triggered by an external signal, so the studio or modular synth itself can be used to control (and sync) the moment of switching to the new preset or patch. Furthermore there are two control voltage inputs that determine the speed of the sequence and that determine the 'id' of the next preset in line (the sequence order).

Since the beginning of this research I also noticed that a matrix of 16 in and 16 out is not really enough to cover all connections in a regular studio setup, but it is a beginning and it will be enough to investigate whether this interface is an interesting new composition-tool.

1.2 **Signal Paths**

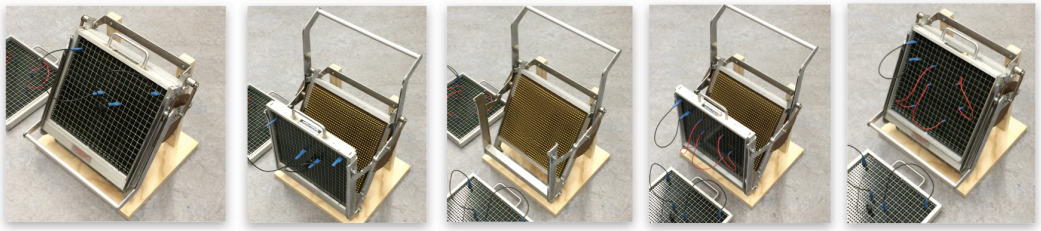
Back in the days the computer was not yet a part of our daily live, telephone connections were manually fixed by people sitting in front of huge panels. From the first moment analogue synthesizers were (commercially) used in the beginning of the 60's, the sounds produced were dependent of the physical connections created with cables. The combination of the different modules that were connected with these cables formed the core of the synthesized sound and this was, and now still is, a very static connection with patch-cables.

Figure 1.2

Once the connections were made and the fundament of the sound generation was found, the limits could be explored only within the boundaries of these hard-wired connections. With big studio setups it could require an effort to re-patch every connection and the patches made had to be well documented. Already back then the time available in an expensive analogue studio was limited and the users wanted to use their time in the most effective way possible. An ingenious system was invented to store and re-call hard wired connections, in order to save precious time. See figure 1.3. The wires were connected to a transportable or portable plate, that could be removed and re-installed so the physical connections were stored and re-called.

Step 1 shows the (very simple) active patch. The storage-device showed in figure 1.3, was integrated in a studio setup attached to the back of the device. In step 2 the lever is moved upwards to detach the plate from the actual patch-panel. Step 3,4 and 5 show how the new patch is moved in place and 'activated' by moving down the lever.

To create connections between two or more points, you need wires. These wires make a physical connection between the output of one device and the input of another. At least that is what was required in the old days and it still holds true today.



Step 1 Step 2 Step 3 Step 4 Step 5

Figure 1.3. Switching between presets, old style.

At the end of the 1960's the EMS company created theVCS3 (Voltage Controlled Studio version 3). The physical cables were replaced by pins and with placing one pin between two points in the matrix (one row and one column) a connection between modules was made. The use of the matrix within these VCS3 synths formed a part of the

inspiration to develop a fully automated matrix version, introducing the possibility to work with dynamic and time-based connections as part of the sound generation and composition of electronic music.

Figure 1.4. The VCS3 Synthy or Putney

1.3 **Pre-research activities**

In 1996 I started working for the Electronics workshop (EWP) at the Royal Conservatoire in The Hague. My electronics mentor and colleague Jo Scherpenisse[2] and I started with the design of one of the first small sensor-interfaces in that time, the Microlab. The Microlab was a small MIDI[3] sensor-interface with 5 analogue inputs and a 7-bit resolution. It could be assembled and built by students themselves and it was cheap enough for them to also purchase. The design was based upon an (for that time), advanced Microchip PIC16F series microcontroller. Being able to convert changing voltages into 7-bit MIDI controller data, was a great step forward for the creation of musical interfaces and interactive installations. Students were now able to build their own MIDI-instrument and control digital processes with signals that originated from the physical world by using sensors.

The next step in the development of new interfaces was the inverse. The possibility to convert digital MIDI data into physical control and actions. For example conversions into CV (Control Voltage), servomotor drive, relay switching, stepper-motor drive, pulse width modulation (PWM) and many more. With this new technology, the students could realize more innovating musical instruments and interactive installations. A lot of the instruments and installations built at that time were based upon this technology — see some examples in figure 1.5.

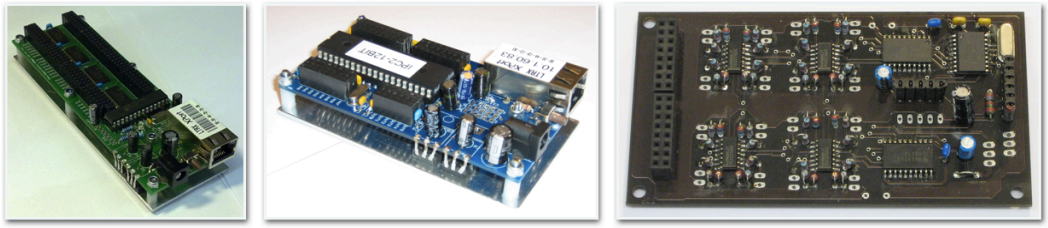


Figure 1.5 Ipson64, IpsonCompact and OSC-CV board

After working with MIDI for several years and teaching students how to apply this technology and after my colleague Jo Scherpenisse retired, I continued with the design of new electronics and the search of new ways for faster communication and higher control resolution. MIDI in that time was and still is limited to a 10-bit (0-1024 steps) resolution and had a relative slow bitrate of 31,25 kbit/s. In my search for new technology, starting around 2004, I encountered Open Sound Control (OSC)[4] as a new musical communication protocol. I also discovered a small embedded web server component, called the XPort. This Xport is a complete embedded web server that communicates using ethernet connections or Wifi. The combination of the Xport and OSC opened up a new world for me to design new OSC-sensor interfaces, like for example the IpsonLab and the IpsonCompact©.

1.4 **From idea to realization and research**

The idea for developing an audio-matrix that can be fully controlled through OpenSoundControl and that can be driven by analogue voltages and external triggers actually arose because of multiple practical requests within the Conservatoire: the introduction of a new mixing console in the Karlheinz Stockhausen-studio[5] and the introduction of new V-FUG's in analogue studio Bea-5 and the refurbished VCS-3 synthesizers with its classic patch-panel matrix. All these subjects, combined with my OSC interface design experience, contributed to the idea of this research project.

1.5 **Stockhausen studio matrix design**

The initial request that motivated me to look into new types of switching circuits, was the purchase of a new mixing console for the Stockhausen studio at the Royal Conservatoire. The old mixer that had to be replaced because of its age and constant growing technical problems, was originally designed and built by technicians of the Institute of Sonology. This audio-mixer was equipped with some very impressive routing capabilities that were not to be found in any other modern mixing console.

It had the possibility of routing a lot of different sources directly to the main faders and from there to the four big JBL loudspeakers. For composers working frequently in the studio it was of great importance that this feature would be restored if a new mixer was to be installed. One of the practical reasons was the position of the audio-mixer, the recorders and the four loudspeakers in the studio. Sometimes it was necessary to listen to the left-

and right-speaker working behind the audio-mixer and in other moments, when controlling the recorders and being turned 90 degrees, two other speakers had to be the left- and right one. With the push of a button the routing of the signals to the speakers could be changed.

The request to re-design this feature in the KHS-studio was the reason for me to look into a solid and more modern switching solution for the first time. I designed a first version of the matrix and re-introduced this unique routing possibility back into the KHS-studio with the use of a small chip called the AD75019 audio switching array (see figure 1.6).

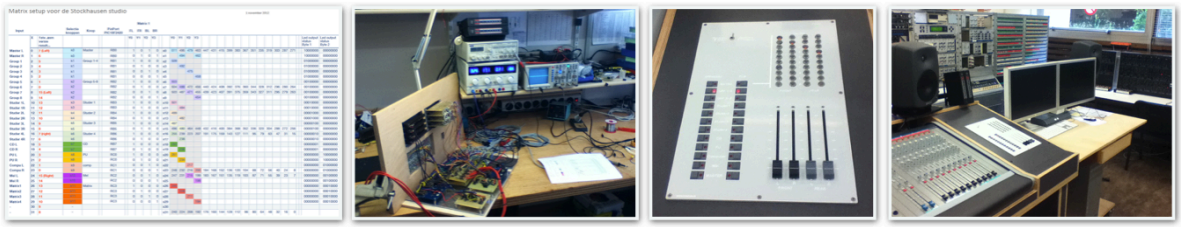


Figure 1.6. Design sheet, electronics and final result of the routing matrix for the KHS-studio.

In the new setup the old matrix-switches still were used to drive the new designed matrix-electronics. Every separate push-button selected a different signal-path from the source, directly to the speakers.

The remote-controlled studio, or RC-studio, forms a side track of my research and actually represents the start of the VC-SPG research. This modular synthesizer-setup consists of different modules in one 19-inch rack which are all inter-connected through the first version of the matrix interface — see figure 1.7.

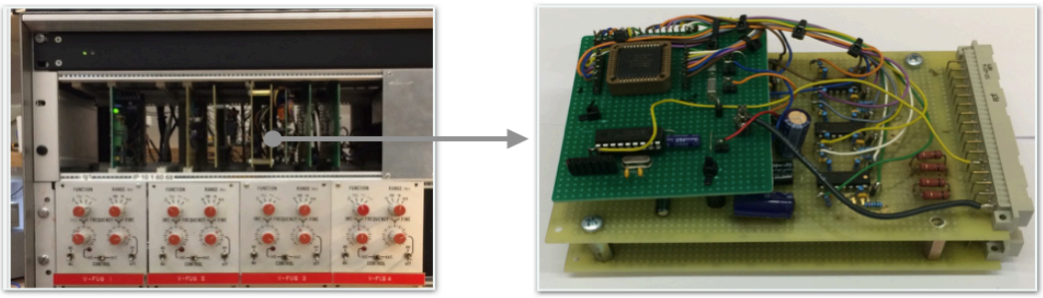


Figure 1.7. The first matrix-board for the RC-studio

The RC-studio is a small analogue modular synthesis system, with 8 x VFUG (Voltage controlled Function Generator), one VOSIM (Voice Simulation), one VSF (Variable State Filter), a matrix and multiple VCA's (Voltage Controlled Amplifiers) that all can be remotely controlled with OSC. The generated audio result is streamed to the internet and

can be listened to by anyone connected to the internet — live, self-played, internet radio.

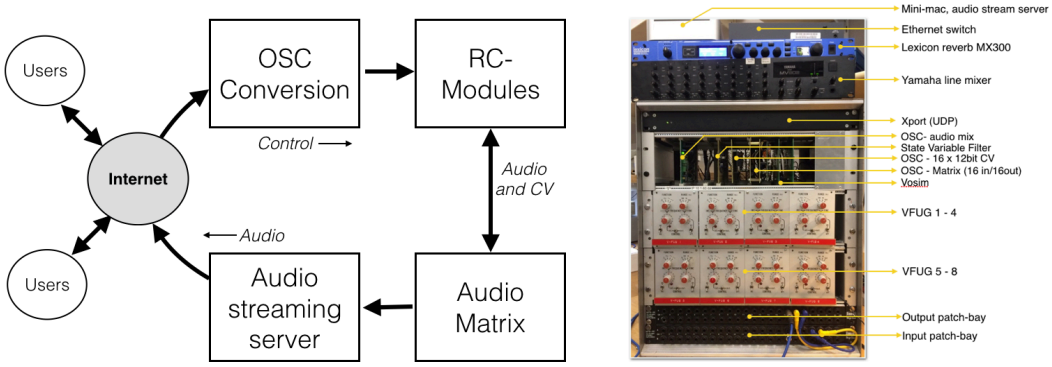


Figure 1.8. RC-studio block diagram (left) and the actual device (right)

The idea to design a RC-studio was realized in small steps and started in 2009 after or during building the matrix for the Stockhausen studio. It started with the installation of 12 brand new VFUG's in analogue studio Bea-5, leaving the 8 old VFUG's unused and waiting to be refurbished, modified and re-used. In that time, I also designed a new 12-bit OSC-CV board and the VFUG's, which have a voltage input to control the frequency, presented themselves to be a great testing object to listen to the pitch-change driven by OSC. This test-setup was the first step in the creation of the RC-studio and it offered me the ideal platform to test new digital to analog conversions.

Of course a remote controlled pitch-change for the VFUG driven by OSC-messages is not enough to fully control the VFUG and the features had to be extended. The next step was controlling the switches located on the front of the modules for selecting the wave shape and frequency range of the VFUG. The switches on the front are therefore bypassed with physical relays and are also controlled by OSC-messages - yet another conversion: OSC to relay control. And since it also is important for any analogue studio to be able to make inter-connections between different modules, the router design already created for the Karlheinz Stockhausen studio, could be used as a remote controlled patch panel. This was the next step in the creation of the RC-studio, the integration of a remote controlled patch-panel, designed with the AD75019 audio matrix switch array Controlled by digital data, this audio matrix can be used to change analogue signals and connections. See figure 1.7.

The AD75019 is a chip that can make hard connections between multiple points. A hard connection means the change of resistance between infinity (no connections and therefor equals the off position) and 150 ohms (equals the on position). The switches are also bi-directional, which indicates the current can flow both ways.

For the implementation of the AD75019 in the design of the matrix, I added operational amplifier (opamp) circuits to the input and the output. The main reason for this choice was

the addition of the creation of the audio- and control voltage summing feature. When multiple inputs are selected to be routed to one output, all these inputs will be added (mixed) correctly by the opamp summation circuit (inverting amplifier).

1.6 **Search for equivalent devices**

During the development of the first model of the VC-SPG, I searched the internet for a device with the same features that I had in mind for the VC-SPG: 16 x audio inputs, 16 x audio outputs, programmable and controllable with the computer and able to work stand-alone, controlled by analogue signals.

The first Google searches for 'OSC matrix' is pointing to my own website (www.ipson.nl), where I describe an older version of a matrix board that can be driven with OSC. After some more extensive search I found a version of an equivalent matrix board, named the Sequential Switch Matrix[6], see figure 1.9. This Eurorack module has 4 inputs and 4 outputs and is manually programmable and does not support OSC. It does have control voltage inputs and the Eurorack module is specially designed to be part of a modular synth only.

Figure 1.9 The sequential switch matrix.

2 **The VC-SPG basics**

The projects described in the introduction, in combination with my expertise of controlling analogue and digital electronic signals, were the key ingredients for the design of the first version of the VC-SPG. The following section covers the main features of the VC-SPG including its routing architecture. Figure 2.1 shows the block-diagram of the complete VC-SPG, designed around the PIC18F2523 microcontroller running the self developed assembly firmware, the AD75019 switching-array and the embedded web-server, called the X-port.

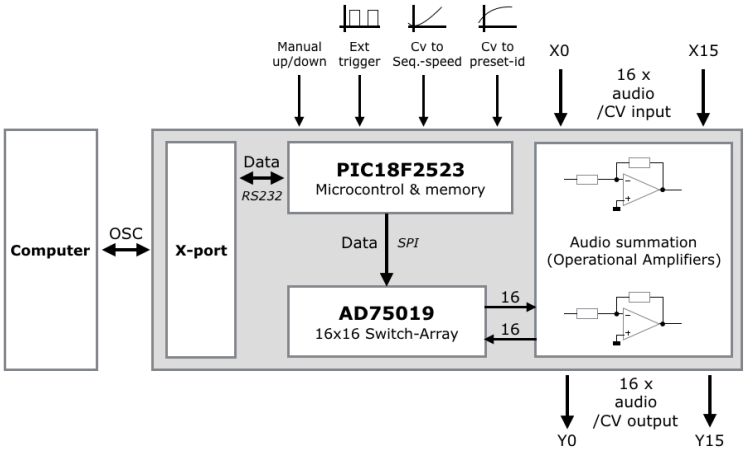


Figure 2.1. The gray block shows the complete interior of the SPG interface.

The fundamental feature of the VC-SPG is the possibility to dynamically control analogue signal-paths (patch cables) with the use of a computer and by means of external voltage-triggers and/or control-voltages. In combination with the self-developed Max/Msp patch (see page 32) the created signal-path presets can be activated sequentially, played-back from a list or the presets can be uploaded to the local memory of the SPG. In total 32 connections can be changed with one trigger or click. The SPG has a local memory that can store up to 32 different presets, all with their own preset-id. If the presets are stored in local memory, it can operate stand-alone and sequence through these presets by itself without the computer connected, making it a real hybrid between the digital- and analogue world. A sequence of steps through the presets, at high or low speeds, can be controlled and synced by external control voltages or triggers. This turns the SPG into a generator, generating sequences of changing physical connections.

The VC-SPG, or CompLex as a few students already nicknamed it, is a programmable and voltage triggered audio-matrix. The SPG has 16 inputs and 16 outputs, all able to handle signals in the audio range between -12V and +12V. The communication is performed with OpenSoundControl (see page 43) over an ethernet connection and the interface has 3 external analog control inputs (see figure 2.2):

1. The external trigger input can be connected to an (audio) pulse changing between 0V and 12V, and whenever this signal changes its value from 0V to 12V (rising edge) or from 12V to 0V (falling edge), the VC-SPG will step to the next preset in local memory. With this input the VC-SPG can be synced with external processes and change presets accordingly.

2. The control voltage to speed (CV-speed) input converts incoming voltage changes to the speed of the sequence. If the CV input is low (0V), the VC-SPG will step through the local stored presets with a low speed. High sequence speeds are achieved with higher values of the CV input.

3. The third input converts the voltage value to a preset-id. Changing the voltage on this input, will also activate the corresponding preset.

The VC-SPG can be used in many different scenarios, but there are two mainstream applications. First the use in combination with a computer connected running software generating OSC-messages and the second application without the computer connected, stand alone. In figure 2.2 a computer is connected and either way it will always initiate with a computer connected due to the fact that an initial set of presets has to be programmed and stored in the VC-SPG.

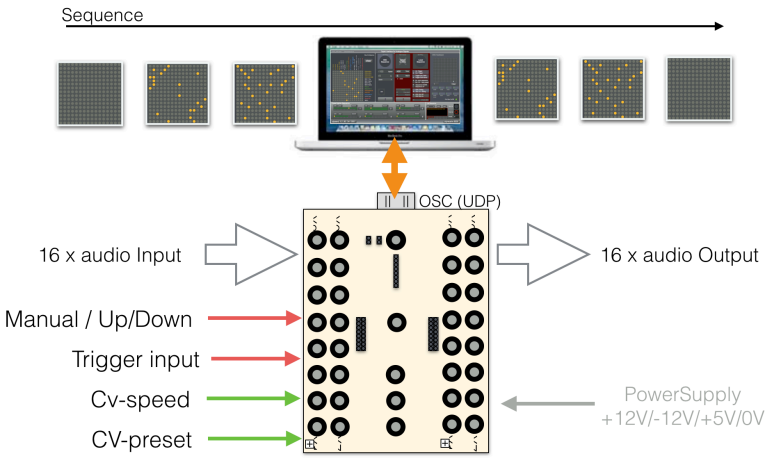


Figure 2.2. Block diagram of the VC-SPG communicating with the Max/Msp patch

The next two chapters 3, Hardware and 4, Software both describe the pure technical design approach of my research project. Since I designed a complete new instrument that consists of hardware and software, it is of importance to share these detailed technical aspects and topics in this thesis. For those readers, who actually do not have any technical background, it may be more suitable to advance to chapter 5, "VC-SPG Features" on page 35.

3 SPG Hardware design

To create a device like the Voltage Controlled Signal Path Generator (VC-SPG), a combination of main components is needed, which will be focussed on in this Hardware section.

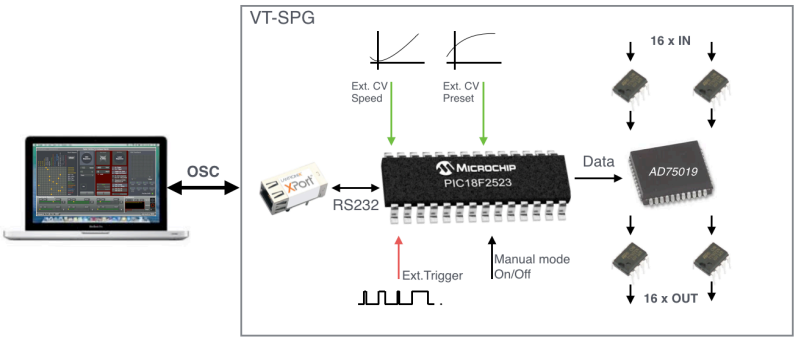


Figure 3.1

First of all the embedded web server, or Xport, as main communication port. This webserver, made by Lantronics, sends and receives UDP and TCP-IP data to and from the network and converts this data into the RS-232 serial data protocol[7] (see communications chapter for more detailed information). The brain of the design is the PIC18F2523 microcontroller which handles all incoming- and outgoing-data, reads the analogue signals and drives the switching process. The actual switching is realized with the Analog Devices AD75019 switching audio matrix.

In this chapter I will focus on the important properties of these main components and explain them. The hardware design of the whole device is somehow generic when you compare it with most of the OSC-devices I already designed: the Xport transmits and receives the OSC-messages and communicates this data-flow with the microcontroller (see figure 3.1). The self-developed assembly software within the microcontroller (also called firmware) determines the functionality and will decide what actions will be taken. The complete data flow and Assembly design will be covered in chapter Software design starting on page 22.

As shown in figure 3.1, the X-port sends and receives OSC-messages and communicates directly with the microcontroller using the RS-232 protocol. Depending on the received OSC-message address-tag and data content, the microcontroller will send the appropriate switch information to the matrix and the audio matrix configuration will be changed accordingly. To be able to drive the SPG with analogue external signals, the microcontroller can also receive control-voltages or trigger-pulses and convert these signals to switching information. An important part of the functionality of the SPG is also determined by the specially developed Max/Msp software patch that can be used to control and program the SPG from a computer.

3.1 **The microcontroller (PIC18F2523)**

"A 28-pin enhanced flash microcontroller with 12bit A/D and nanoWatt technology"[8]. That's the main title of the Microchip data-sheet or user-manual, containing 390 pages full of detailed information. Because the PIC18F2523 is the main component of the interface design, it is of big importance to explain some of the main features of this controller. The PIC18F2523 package is a 28-pin DIP (dual in line) chip that has 28 physical pins to be connected. Like most chips, or active components, this controller needs two pins for the power connection +5V (VCC) and 0V (GND).

Pin 1 of the controller, see figure 2, is the master-clear pin. When this pin is connected to 0V (GND), the program running inside the microcontroller, also referred to as the firmware, will reset the program running inside and it will restart from the top.

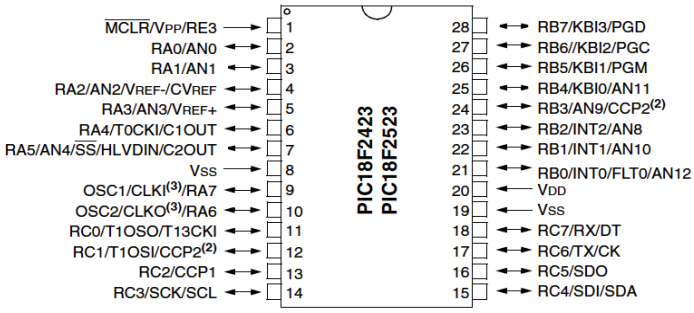


Figure 3.2. The pin-out of the PIC18F2523 and the package.

Check appendix A for a complete block diagram of the PIC18F2523 microcontroller. Like any microcontroller, almost all the pins are inputs and/or outputs, clustered in ports. This particular 28 pin controller has three of those 8-bit ports, called porta, portb and portc. The pin names referring to these ports start with a capital R, followed by A, B or C (see figure 2). All ports are 8-bit wide which means portb for example consists of portb,0 (=RB0) until portb,7 (=RB7), making 8 bits in total. All communication with the peripherals (the X-port, the audio matrix, the switches) is realized through these 3 main ports. Within the text and the figures in this document, the port-numbers are also referred to as RB0 (= portb,0) or RA5 (=porta,5).

As part of the multifunctional design of the microcontroller, multiple functions are gathered in one pin. Depending on the configuration of the microcontroller or initialization part of the process, the pins will have a specific function. This all starts with the definition of inputs and outputs within. All three 8-bit ports can be inputs and/or outputs within the same application. The definition of inputs and outputs within the design of the interface is a puzzle and often the choice of the type of microcontroller depends on how many inputs and outputs it has.

See figure3.3). The colours are used to divide the signals into different categories.

Green is input for the controller, these can be control-voltages or switches. Yellow is data communication using the SPI protocol (Serial Peripheral Interface), where pin 11, 12 and 13 transfer the data to the audio switching array and pin 26, 27 and 28 are communicating the preset numbers in manual mode to the 5 external led's. The light-blue colour indicated the connection with the X-port, communicating RS-232 with the processor (Rx = receiving and Tx = transmitting). The black and red colours are used for the power supply and one dark blue connection is used to indicate the master clear (MCLR) pin. In appendix B this same figure with more detailed information about all individual pins can be consulted.

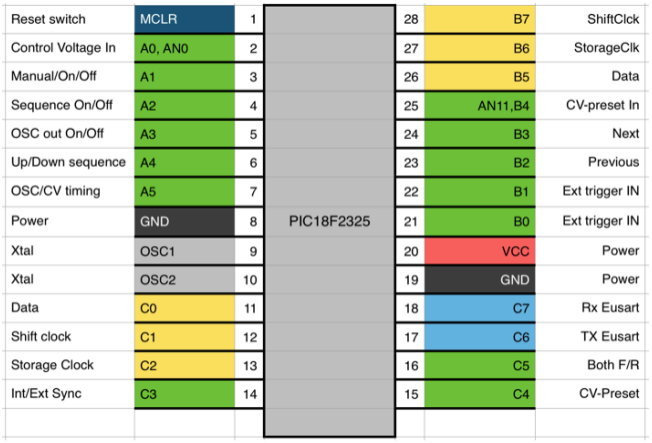


Figure 3.3. The pinout of the microcontroller with SPG functionality.

3.2 **AD75019 16 x 16 Crosspoint Switch Array**

The main component of the VC-SPG that does the physical switching is the AD75019, made by Analog Devices. This chip contains 256 analogue switches in a 16 x 16 array (see figure 3.4. The switches are bi-directional, which means either the X or the Y can be input or output. Because the design of the chip is based on CMOS[9], the resistance of the switches in 'on' position is 150 Ohms and in 'off" position the value is infinite. The data-sheet of the chip describes this: "the AD75019 is fabricated in Analog Devices' BiMOS II process. This epitaxial BiCMOS process features CMOS devices for low distortion switches and bipolar devices for ESD protection". In the design of the VC-SPG the direction of the signal is determined by the additional operational amplifiers (opamps) which function like an audio buffer and audio mixer and it defines the input and output.

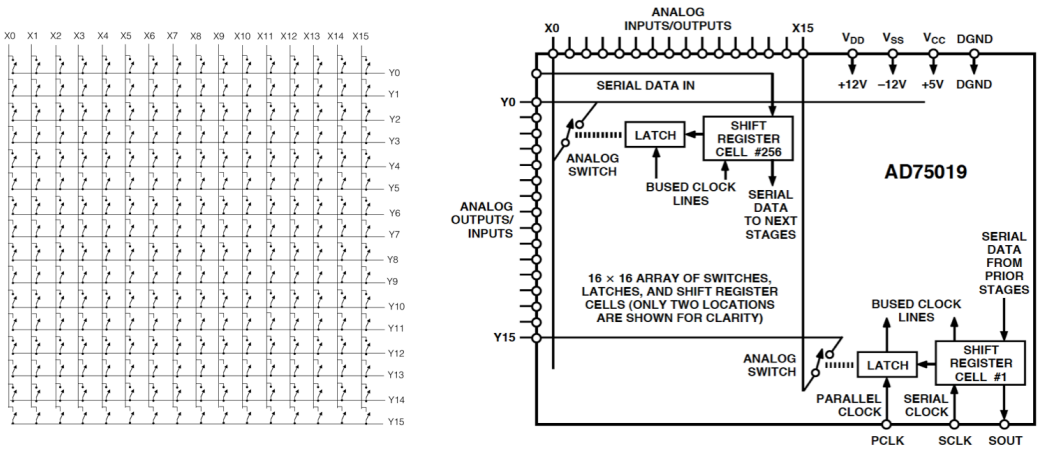


Figure 3.4. 256 switches in a matrix and the functional block diagram of the AD75019.

Besides the power connections of -12V, +12V, +5V and GND the AD75019 has three connections for the data communication: the PCLK, SCLK and SIN. The method of transferring the switch information from the processor to the array is also referred to as SPI or I2C. In the chapter 'SPG communication' (page 43) this method is explained in more detail. SIN is the serial data input and all data passes through this input. The (switch) data is 'clocked in' with the SCLK. This is a pulse signal changing from 0V to 5V at a high frequency rate. Every time this signal changes from 0V to 5V, also called the rising edge, one bit is clocked into the onboard data register. Per writing cycle for the whole array, this has to be done exactly 256 times. After all bits have been 'clocked in' into the data-register, the new status of the switches has to be activated. This is done by changing the PCLK signal, which on a falling-edge (5V to 0V), will store and activate the new switch configuration.

3.3 **The Xport**

The Lantronics X-port is an embedded ethernet device server and is designed to add serial capability to any electronic device that has serial communication available. The Xport can receive and send UDP/TCP-IP packages from the internet and convert this to the serial communication protocol RS-232. The component looks like a single RJ45 connector (figure 3.5), but on the inside a complete 32-bit processor is running to process all incoming and outgoing data.

Figure 3.5. The X-port, a complete embedded ethernet device server.

The Xport has a RS-232 input and output which within the configuration of the SPG both are directly connected to the Tx and Rx ports of the microcontroller. The OSC-messages that are send from the computer to the SPG use UDP (User Datagram Protocol) to travel over the net. UDP is a standard data protocol. When the OSC-message is received by the X-port, the data is converted to RS-232 and transmitted to the microcontroller.

To configure the X-port and tweak its settings a standard web browser is used. During the building process of the SPG, the Xport is already configured and tweaked to be the perfect communication partner for the microcontroller. The only variable that has to be set if a new user is connected to the SPG, is the remote IP-address (this is the IP-address of the user).

3.4 Operational Amplifiers (Opamps).

When audio signals are being patched, mixed or switched it is almost always realized with the use of operational amplifiers, or opamps. An operational amplifier is an ideal building block for all kind of electronic applications, but especially for handling audio-signals.

An opamp has 3 important properties, that make it ideal for electronics. (1) It has a very high input impedance which causes the current flowing into the opamp to be almost zero. (2) The amount of amplification between in- and output is possibly infinite . Without feedback compensation the output of the opamp will always hit the positive- or negative power-supply rails. (3) It has a very low output impedance, which allows the opamp to have a strong output current. The chapter 'Features' (see page 35) already focussed and describes the use of the opamps for the input and the output of the matrix. The circuits used for both input and output are called 'inverting amplifiers' and are one of the most common applied mixing circuit designs.

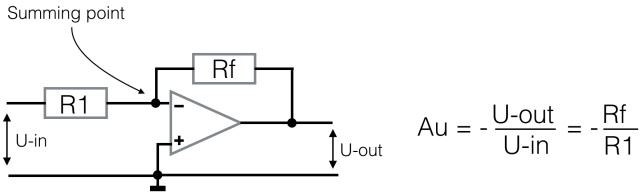


Figure 3.6. Inverting amplifier circuit.

Figure 3.6 shows an inverting amplifier with Rf (feedback) and R1. The ratio of these resistors determine the amount of amplification, and in case of the in-and output of the SPG, both resistors are 10kOhm - the amplification is -1. If in case of the SPG multiple inputs are selected to be routed to one output, a summing amplifier will be created, like shown in figure 3.7.

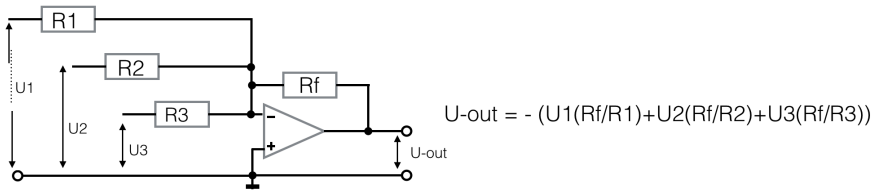


Figure 3.7. Inverting amplifier with multiple inputs and U-out is the sum of all inputs.

As already highlighted in chapter 'Features', connecting multiple inputs to one output is creating a perfect addition (mix) of the input signals on the output. If one input-signal is selected to be routed to multiple outputs, the inputs signal has to be split .With the configuration the SPG has currently, the amplitude of the input-signal will decrease every time the signal is split. Every time an input signal is split, multiple negative inputs of different opamps are tied together, confusing the amplifier circuit setup. There are solutions to this problem, but at this point this 'feature' is still not fully implemented.

3.5 **CV- and Trigger-input adjustment**

The SPG contains three external inputs that have to be conditioned before the signal can be fed to the input of the microcontroller. The microcontroller is so called TTL-compatible. This means the inputs of the controller cannot exceed voltages higher than +5V. A standard euro-rack modular system uses audio signals that vary between -5V and +5V. The pulses and triggers of this system are in between 0V and +12V (measured with the Doepfer A-100 system).

These values are too high for the microcontroller to process, and therefore these signals have to be conditioned. The variation from -5V to +5V has to be converted into 0V - +5V and the pulses have to be lowered from +12V down to +5V.

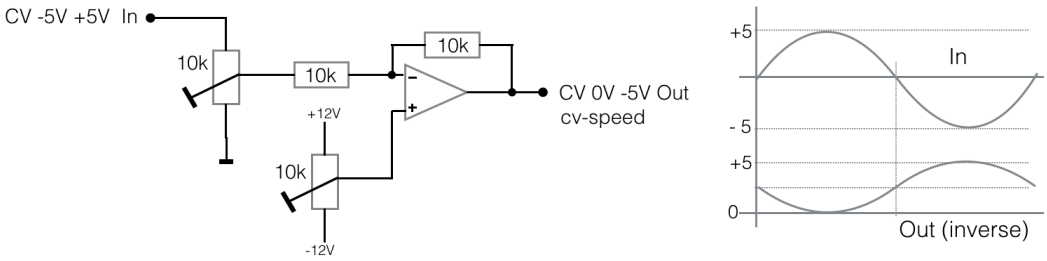


Figure 3.8. Conversion of the cv-speed is inverted.

The conversion from -5V to +5V is realized with an opamp circuit as shown in figure 3.8. The input control voltage (from the euro-rack system) is first divided by two with the use of a 10k potentiometer connected to the input. Because the input signal is connected to the negative input of the opamp(-), the output will be inverted (inverting-amplifier). Unlike the conversion of the control-voltage for the preset-id, this control-voltage has to

be converted due to the use of the onboard timer where a small number (= a low value) results in high frequency. The trimmer potentiometer on the positive input creates an offset voltage, so the new 'zero-crossing' on the output is lifted to 2,5V as shown in the graph. A similar circuit is used for the conversion of the CV-preset-id, except this conversion does not have to be inverted.

The external trigger pulse coming from the euro-rack modular is switching between 0V and +12V. This has to be converted in to 0V and +5V. There are multiple ways to realize this conversion, for example with the simple use of an avalanche-diode. An avalanche-diode has a certain threshold voltage after which it starts to experience an 'avalanche breakdown'. In other words, if the voltage parallel to the +5V-avalanche diode exceeds +5V, the voltage will not be higher than 5V because of the avalanche effect. The diode acts as a compressor.

In the modular version of the SPG the solution of an opamp was used. If an opamp has an applied power of +5V, the output will never exceed this 5V, because that's the maximum it can deliver. If +12V is applied to the input, the output will be +5V. Take a look at figure 3.9.

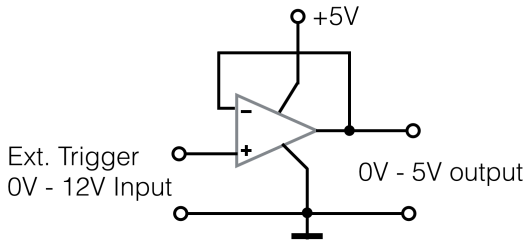


Figure 3.9. Follower circuit with opamp powered with +5V.

3.6 **SPG circuit layout**

Below in figure 3.10 the complete circuit layout is shown. A more detailed and extracted version of the same picture can be found in appendix C. In the process of designing a new device, drawing the circuit layout is not only very important for document reasons, but it is also the fundament for drawing the Printed Circuit board.

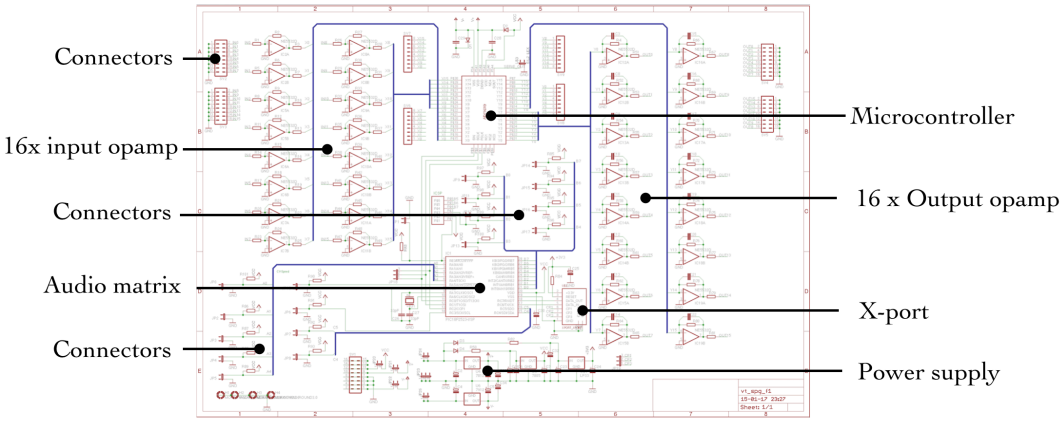


Figure 3.10 Full schematic of the VC-SPG (minus the additional modifications)

Drawn with Eagle software, the circuit-layout shows all the main components, some smaller components and the power-supply all inter-connected to the right pins. There is a difference between the thin red wires and the thicker blue wires used in the circuit-layout. The thicker blue wires are so called 'busses'. They consist of multiple wires going to and coming from the same component. Busses are normally used to route address-lines or data-lines. Using these multi-wire busses in a layout creates a clearer view of the circuit instead of drawing every wire separately, like the red-ones. The biggest surface of the circuit is used for the 32 operational amplifiers, 16 times input and 16 times output (see page 35).

The one part that is not drawn in this circuit layout, but will be added an updated version, is the adjustment of the control voltage from -5V/+5V into 0V/5V and the adjustment of pulses from 0V/12V to 0V/5V. This part of the circuit is added in a later stage and will be part of a modification for the next model.

A very important and obvious part is the power-supply section. The SPG uses three intern voltage regulators to create +12V, -12V, 5V and 3,3V. The use of +12V and -12V (balanced power-supply) is a common way of powering operational amplifiers and audio electronics in general. Audio-signals, or AC- signals, move from positive- to negative values and if these signals have to be modified (amplified, filtered, muted, etc …) a broad dynamic workspace is needed to realize this. Hence the balanced power-supply.

The +5V, a part of the external power adapter as well, is used for the digital electronics inside the SPG. It powers the microcontroller and the digital part of the audio-matrix. The 3,3V, extracted from the +5V with a LF33 voltage regulator, feeds the current consuming Xport. The Xport needs quite some current to operate, around 200mA in quiescent state.

3.7 **Printed Circuit Board**

To make production of the SPG easier and faster and to offer the students the

possibility to build and construct one, it's a must to design a Printed Circuit Board (pcb). On a pcb all connections are pre-made according to the schematic and the only thing the user has to do is place the components on the right location and solder the pins to the board. The pcb, drawn with EagleCad software, is a straight translation from the circuit-layout. It is a double sided pcb, which means that both bottom and top layer consists of copper connections.

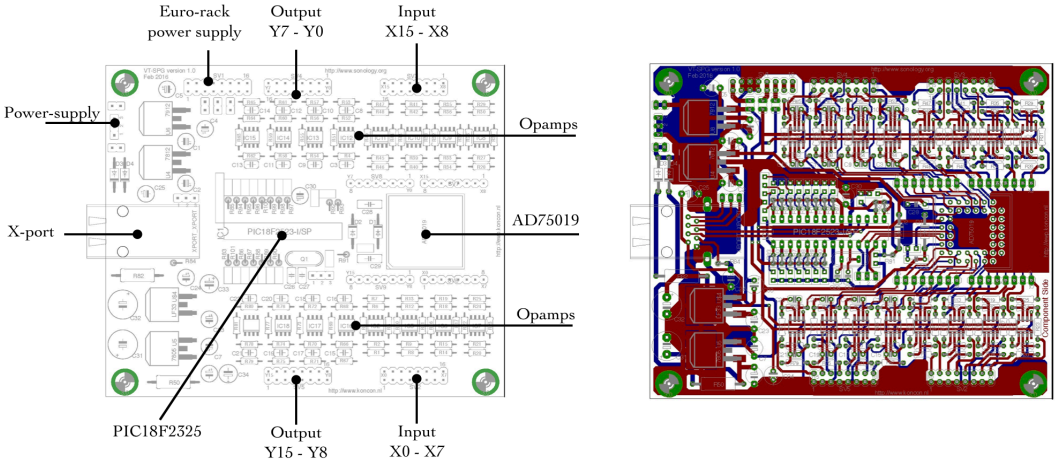


Figure 3.11. SPG printed circuit board.

The left side of figure 3.11 shows the component side, with only the names and the shapes of the components printed on top. The right side of the figure shows both the top (red) and the bottom(blue) copper layers. The big surfaces are ground planes and are used to keep the ground impedance as low as possible (more copper) and it also has environmental reasons - the less copper that is removed, the better.

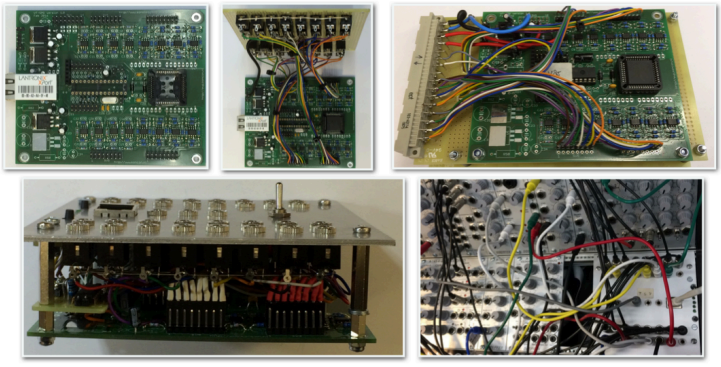


Figure 3.12. SPG printed circuit boards wired up

3.8 **The SPG prototypes**

To be able to test the hardware design of the SPG and have students practically work with the instrument and share their feedback, I made multiple models to be tested. Since there are multiple standards concerning the connectors in the analogue synthesizer world, the models are provided with different kind of connectors and with different shapes. A

quick reference list of all the models built, can be found in appendix D, page 70.

The patch-panel of analogue studio Bea-5 is designed with so called banana connectors. This is a robust connector with only one signal present. The setup of the analogue studio has one collective GND connection for all the modules within the studio. If connections have to be made between these modules, only the signal has to be patched - the GND connection is already made. Figure 3.13 on the left shows the banana-connectors and patch-cables and in the middle a small part of the patch-panel from analogue studio Bea-5. On the right side the SPG prototype with female banana connectors, to be interfaced with the patch-panel of Bea-5.

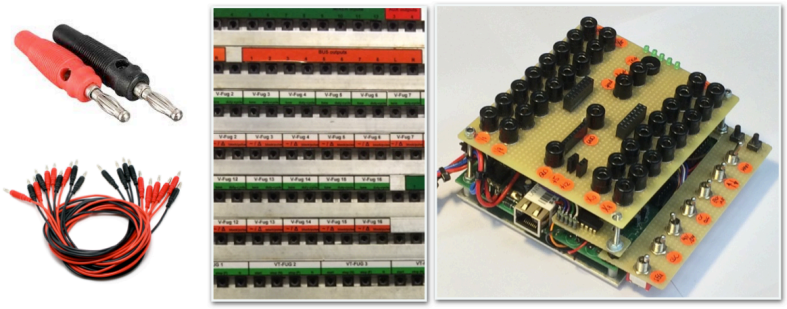


Figure 3.13 Banana connector model

The next implementation of the SPG is the model that communicates with mono mini-jack connectors. These small connectors are used in a wide variety of modular systems and especially in the Eurorack modular synths, like for example the Doepfer A-100 system. The minijack connectors connect two signals: the GND and the actual signal.

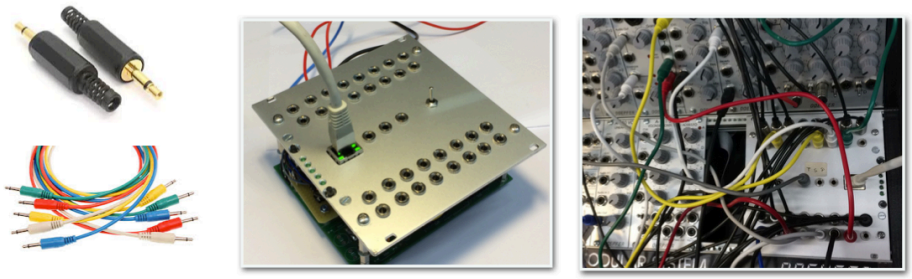


Figure 3.14 On the right, the SPG model with minijack connectors

The model shown in the middle in figure 3.14 is the SPG model that can be build into an Eurorack modular synth. At this point two of these models have been build: one version for the Sonology Doepfer A100 system and one version for Ruben Brovida, who embedded the model into his own modular setup.

The third type of SPG prototype is provided with the 1/4" mono jack. This is a popular connector in the music industry, especially for guitars and keyboards, effects, amplifiers and mixing consoles, see figure 3.15. Due to the size of these connectors, the package of

this model with 35 female 1/4" jack connectors mounted, is somewhat big.

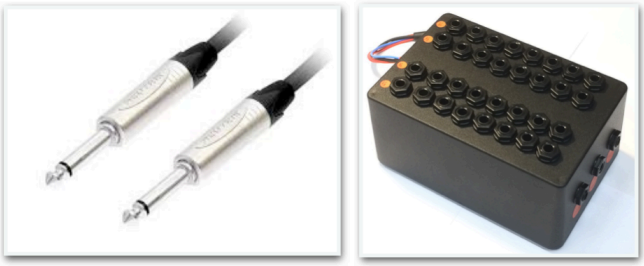


Figure 3.15

The last practical model of the SPG that is built within the timespan of the research-project, is the version that is implemented into the RC-studio and is constructed on a Eurocard format pcb (printed circuit board). The inputs and outputs, as well as the power-supply, is provided by two 32-pin multi connectors - see figure 3.16.

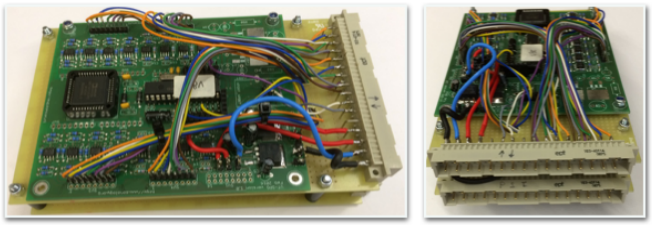


Figure 3.16. The VC-SPG model for the RC-studio

4 Software design

The core of the SPG project is the software and it can be split in two different main applications. The first one is the assembly code that is running inside the microcontroller, processing all the analogue- and digital-signals. This code running inside the Microchip microcontroller is also known as firmware. The second software application is the Max/Msp-patch running on the host computer, driving and programming the SPG (see figure 4.1).

The Max/Msp patch generates a stream of OSC-messages that drives, programs and controls the SPG. These OSC-messages, explained in chapter 'Communication', page 43, can be generated by any type of software, as long as the format of the OSC-message has the correct syntax. The firmware, however, is the heart of the design and the setup and configuration determine the main physical properties of the SPG. Choices made within the firmware have time-related consequences and thus, have a big influence on the musical behavior of the SPG.

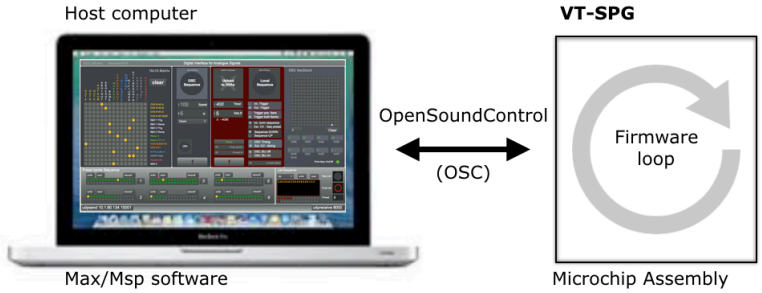


Figure 4.1 Max/Msp patch and Assembly determine the properties of the SPG

4.1 Assembly- firmware

It is inevitable to explain something about the programming language Assembly. The firmware written for the SPG consists of around 3000 lines of code - a huge part of the projects' development time and important parts of the properties of the SPG are determined by this firmware and its configuration. It would be beyond the scope of this thesis to cover the complete architecture of microcontroller and the Assembly language in total, but a glimpse at the technology used is essential for this research project to give a better insight and understanding.

Assembly is a low level computer programming language that is specified for certain types of controllers. An Assembly program written for one specific device or brand, cannot easily be re-programmed into another device due to the architecture differences between the devices. Also the code and the structure of the program very much depends on the architecture of the controller.

The PIC18F2523 microcontroller, an 'enhanced flash microcontroller with 12 bit A/D nano watt technology' (Microchip user-manual), is an 8-bit RISC-controller[10] with powerful features. This microcontroller is, in fact, a small computer complete with inputs,

outputs and power connections. Instead of the normal mouse and monitor, a microcontroller has physical pins with voltages between 0 and 5V acting as inputs and outputs. In visual aspect it does not look like a computer at all. It is a chip, an Integrated Circuit or abbreviated as IC. It has program memory, data memory, multiple timers and counters, communication ports and special function registers.

Writing firmware for this controller means writing the program code on bit-level, the lowest programming level. This actually means the programmer (that would be me) has to literally think in 'bits' and 'bytes', thinking in time slices of 20 nanoseconds or having to realize that over 5 million steps of code will be executed in 1 second.

Another subject that has to be indicated, is the importance of the registers inside the microcontroller. The special function registers (SFR) within the microcontroller, contain data or information, which determines the way the microcontroller operates in realtime. At startup or initialization, all registers are filled with data (8-bit or 16-bit words). Therefore registers can be compared with switches that have to be set or cleared to create a certain configuration. The main and most important register in the controller itself is the working register called Wreg. All commands, operations and actions will be processed through this register Wreg. Moving numbers from one place to another, adding numbers, shifting numbers, comparing variables, (almost) nothing can be accomplished without Wreg.

Another important register is the status-register. In this status-register the actual 'state' of the controller can be checked after an operation or arithmetic action. For instance if two numbers are added and the outcome is a number larger than 255 (8-bit number maximum), the corresponding carry-bit within the status-register will be set and can then be checked by the next instruction.

To give a rough overview, the Assembly language instructions can be divided into four big basic categories: byte-oriented operations, bit-oriented operations, literal operations, and control operations. Every operation or instruction consists of three operands. Some instruction examples of the 4 different categories:

Examples of byte oriented operations / instructions:

andwf	**porta**	[AND the content of the Wreg with register porta]
incf	**counter**	[increment variable counter with 1]

Examples of bit oriented operations:

bcf	**porta, 1**	[bit clear f // clear bit 1 of register porta]
btfss	**porta, 3**	[bit test f and skip if set // test bit 3 of porta and skip the next instruction if this bit is set]

Examples of literal oriented operations / instructions:

movlw	**0x00**	[move literal into Wreg // move hex number 0x00 into Wreg]

xorlw 0x2A [Xor literal with Wreg // Exclusive OR content of Wreg with 0x02A]

Examples of control instructions:

call wait [call subroutine wait]

goto write [goto subroutine write]

With regards to the design of the SPG, many reasons can be given for using Assembly and a fast microcontroller. First of all, my familiarity is based on the usage of the Microchip controller series, given the fact that I have designed extensively with this technology. I'm experienced with the design of electronics in combination with the microcontrollers running Assembly. This experience gives me the possibility to make a better design in the time window available for this research - I do not have to learn a complete new language.

The second reason, and the most important one, is that being able to program Assembly is a very powerful way to maintain control over 'time' during operation. This means the time elapsed in an Assembly program can easily be managed and calculated. Every instruction only takes one or two instruction cycles (clock cycles), from which the time-length is known. Creating some loops or equivalent actions can always be brought back to the actual real time elapsed within that loop.

Higher level programming languages like C++, Java, Pascal or Arduino's 'processing' need to be compiled with external compilers, bringing the program back (down) to machine language. This translation of the written program with the aid of a third party compiler is a conversion or translation in which this exact level of time control is lost. Depending on the quality or brand of the compiler used, the actual generated machine code can differ and so will the elapsed time of different loops and routines. When depending on fast time-critical communication, it is important and absolutely necessary to keep control over 'time' in the written code. This control can be obtained when programming in Assembly.

The best way to explain the way Assembly works is by using an example. In the next paragraph follows more detailed information about how the switching array can be filled with a new preset and activated - the core activity of the SPG.

4.2 **Data transfer from microcontroller to matrix**

The core assembly process in this project is to drive the Analog Devices AD75019 switching array chip. This 16 x16 matrix chip is capable of switching 256 switches simultaneously. Changing the status of these switches takes time. It takes the controller 568µsec (= 568 x 10^{-6} seconds) to write 256 bits from the controller output into the

switching array. In other words, within 1 second, this switching array can jump 1700 times between different presets (1.7kHz).

Another important process in the setup is reading the incoming opensoundcontrol (OSC) information from the computer connected to the controller. Through external software (for example Max/Msp or Super Collider), the controller can receive up to 32 presets, to be stored into the local memory and later to be sent to the switching array when the sequencing-mode is active. Or the received OSC-message contains a new patch that will be activated immediately after it has been received.

How can an eight bit controller like the 18F2523 drive the AD75019 and simultaneously read incoming data from the web server or computer? Throughout this research project it has become more clear that speed and timing matter. The speed of switching between presets is an important variable when the SPG is used as a generator. In this chapter the importance of this speed issue will be highlighted by describing the complete process of transferring data between the controller and the switching array. This is the most essential process of the whole generator.

Before the time related process can be explained, it is important to know how the controller is communicating with a peripheral. Take a look at the timing diagram below, figure 4.2. Peripherals like the AD75019 switching array do have 3 data communication lines: data input, a clock input or shift clock (Sclk) and a storage clock (Pclk). To write a full preset from the controller to the switching array, 256 bits have to be transferred one by one with the aid of these 3 lines only. See also chapter 6 'SPG communication' for the exact syntax of the data.

Every bit represents one switch within the array (1=open and 0=closed). The first bit that will be 'clocked' into the array is in fact the last bit of the matrix: (x15,y15). After 256 clocks or positive edges, the first bit (x0,y0) is set into place. If all bits are in place or 'clocked-in' the storage-clock signal will change value. This will result into an activation of the preset itself and all switches change to their new position simultaneously.

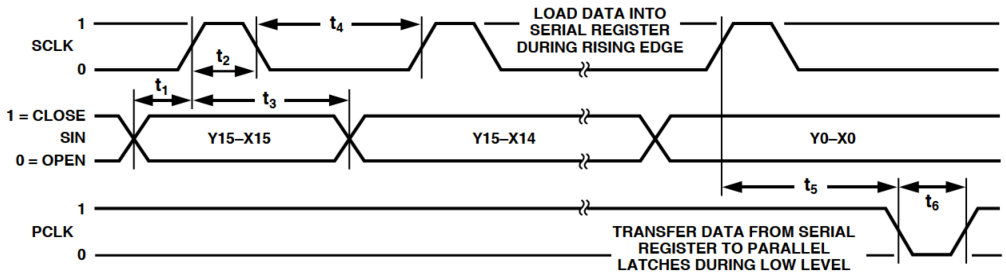


Figure 4.2. Timing diagram of communication (SPI, Serial Peripheral Interface)

The clock-signal, SCLK, is generated by port RC1 of the controller and switches between 0V and 5V at a very high speed. Within the Assembly code port RC1 is set and

reset sequentially, to generate a clock-pulse. On a rising edge, the value of the data input SIN will be 'clocked in'. The clock, the data and the storage clock are generated by the assembly program, running within the controller. Port RC0 is the data output from the controller, connected to the SIN (data input) of the array. Port RC2 is the PCLK - the most important signal. When the PCLK is set low, the whole preset is activated.

It is now easier to look at the corresponding Assembly code that realizes this communication. The name of the routine in the example below is called 'WriteData'. Within the main program of the generator, this part of the code is named a sub-routine. This sub-routine is 'called' from within the main routine with the value to be sent to the switching array stored in the variable 'Waarde'. So, if a valid value has to be written to the AD75019, this routine 'WriteData' is executed. The routine only writes 8 bits in a row, hence the value 8 at the beginning of the code. For writing one complete preset from the processor to the AD75019 switching array, this particular piece of code will be called 32 times (8x32=256). The flowchart on the right shows the sub-routine in graphical format.

```
WriteData
   movlw     d'8'
   movwf     bitcnt
write_x
   rrcf      Waarde
   bc        write_0
   bcf       portc,0
   goto      write_1
write_0
   bsf       portc,0
   write_1
   nop
   bsf       portc,1
   nop
   bcf       portc,1
   decfsz    bitcnt
   goto      write_x
   return
```

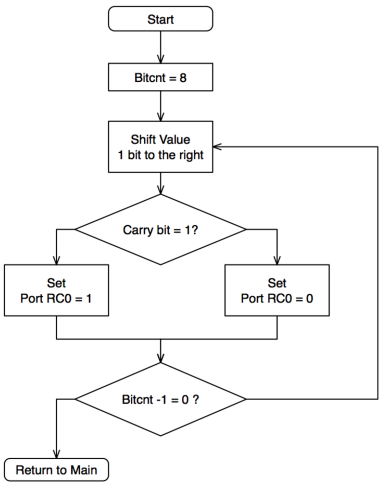


Figure 4.3. Flowchart with the corresponding Assembly code.

During the first two instructions the counter is set to the value of eight - only 8-bits will be transferred in this routine. First loading the Wreg with the decimal value 8 (= movlw d'8' // move literal into Wreg) and then moving the value of the Wreg to the variable 'bitcnt' (= movwf bitcnt // move content Wreg to bitcnt).

To be able to determine the value of a particular bit, it is best to just rotate (or shift) the value 1 bit to the right (rrcf waarde // rotate right through carry f) through the carry. If the result of this action sets the status registers carry bit, the data port RC0 can be set to one (= bsf port,0 // bit set f). If the carry bit is cleared, port RC0 will be set to zero (= bcf portc,0 // bit clear f). If the data pin, or port RC0 has its right value, the Sclk pin (port RC1) will be set high and low, thus creating a rising edge. To make this clock pulse longer in time, due to minimum values, a nop instruction is placed in between (nop = no

operation).

The last decision in this sub-routine is to check if all 8-bits have been dealt with. Decrement the counter value with one (bitcnt-1) and check if the resulting value is zero. As stated before in this chapter, the time that is needed for writing the whole preset from the controller to the switching array, can now be determined accurately by calculating exactly how many instructions it takes. In practice, a good quality oscilloscope can also provide precise information about the length of the whole process - receiving the OSC-message from the computer, storing it and sending it to the switching-array.

The last signal that has to be generated in order to really activate the 256 bits, is the Pclk (see figure 4.3). In the design of the SPG this Pclk signal is connected with port RC2 of the microcontroller (figure 4.3). After writing 256 bits, port RC2, which is normally high, has to be cleared and set in sequence to provide the AD75019 with a falling edge. If the falling edge is detected, the preset with all of its 256 bits is activated and the SPG switches. Within the design of the SPG, this negative pulse on RC2 is actually one of the most important timing related pulses. Within the Assembly code this routine is called 'metro'.

4.3 **The SPG main-routine**

The heart of the assembly code of the SPG is the main-routine. This routine is actually a loop containing jumps and calls to different sub-routines, continuously checking if something internal or external has changed. It is a continuous process and will only stop if the power is switched off. Also notice that the main program-loop consists of checking exclusively the state of 'bits', resulting in questions that only can be answered with Yes or No.

Consult the flowchart in figure 4.4. Starting from the top, the first process to be checked is if new valid OSC-information is available. If the answer is no, the next status to be checked is the manual mode. Is the manual-mode switch active? If this is not the case, the last status to be checked is the sequence mode and if that question is also answered with No, the whole loop starts all over again.

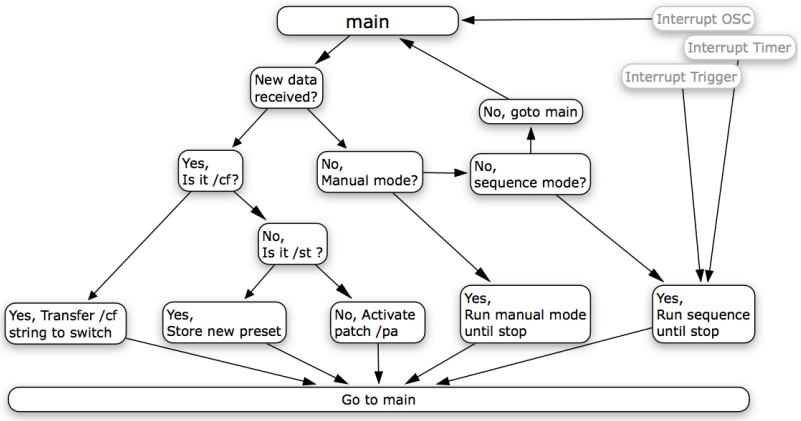


Figure 4.4 The SPG main-flowchart.

If new data is available (new data received = yes), there are 3 different OSC-address type tags to be checked. If it is a /cf message (configuration), then the status of the SPG has to change - it has to for example start with sequencing or store new values for timing. Should it pertain to a /st (store) message, the received patch has to be stored in the right memory location. Lastly, if the block in the middle is dealing with a /pa message, this patch has to immediately be made active.

The 3 light-gray circles on the top/right are an indication of the interrupt processes that can occur. These interrupts are triggered by external processes (trigger pulses) or internal processes (timer counters).

4.4 External and internal interrupts

An important advantage of the microcontroller is the use of interrupts. Like in normal life, every process can be interrupted. The question is how to handle this situation and which next process will follow up this interrupt. Can you proceed with the same subject, when somebody just interrupted your 'actual action' or do you ignore the interrupt? The trick is to store the 'actual action' just before you react on the interrupt and deal with it. All these actions in a sequence within Assembly are called an 'interrupt handling routine'.

The microcontroller has many different kinds of interrupts. There are internal- and external-interrupts, high-priority and low-priority interrupts and all of them have to be configured during initialization.

The design of the SPG uses a few important interrupts, which will be explained in this paragraph. The functionality of the SPG is based upon 3 different kinds of interrupts and they all have a high priority - they have to be taken care of right away. These interrupts do really 'interrupt' the main program loop. Figure 4.5 shows the block diagram of the main routine in combination with the interrupts.

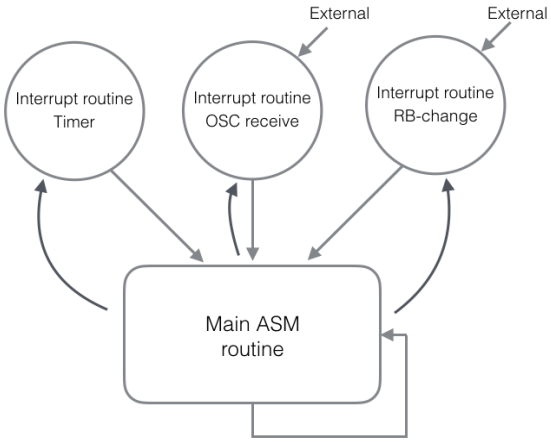


Figure 4.5 Main routine with 3 types of interrupt.

The first interrupt is the reception of new OSC-messages. When new data arrives, it has to be processed right away. The second one is the internal timer interrupt. After a certain amount of time, mainly used during sequencing, the timer sets an interrupt, stating the adjusted time has passed. The third interrupt is the so called 'interrupt on change'. If a signal connected to a particular pin changes value from low to high (0V-5V), or high to low (5V-0V), it will generate an interrupt. This 'interrupt on change' is an ideal way to time-sync the SPG with external processes.

4.4.1 Receive OSC-message interrupt

This interrupt occurs when a new OSC-message is received and it activates the EUSART[11] RCIF flag. This flag, or bit, is set high when new valid data is received and is waiting to be processed. The processing of the received data is done within the interrupt

routine, after the values from the main routine are being put to stack.

As you probably know, an OSC-message is build up by three different tags, starting with the address tag, the first thing that has to be checked within the routine is the address tag. Is it the right value and is the data intended for the SPG? The values intended for the SPG are: /pa, /st and /cf (see chapter OSC-communication) and all three call for different actions within the SPG (activate patch, store patch, control SPG). As a result not only the address-tag has to be checked, but also the linked process has to be determined. The flowchart or graphical representation of the whole (complex) interrupt routine can be found in appendix E.

The OSC-message 'receive interrupt-routine' will handle the interrupt and sets the corresponding bits for the main routine to work with. If a /pa OSC-message has been received, the main routine should activate the corresponding patch right away. If the /st OSC-message is received it should store the preset at the right memory location and if the OSC-message has a /cf address tag, the SPG will change its present state.

4.4.2 **Timer interrupt**

When the SPG is sequencing it makes use of the internal timer (timer0). A timer is a 16-bit register that counts every program step within the code. It actually increases its value with 'one' on every instruction (cycle). When the /cf OSC-message contains a 16-bit timer value that is the source of the sequence speed, this number will be compared with the value of the internal timer. When both are equal an interrupt will occur and the SPG will step to the next preset stored in its memory.

Hence sequencing with the SPG is done in sync with its internal timer and this results in a very stable and precise timing, which for musical applications, is critical.

Also in the CV-speed mode, when the SPG is reading an external voltage to be converted into a sequence speed, the internal timer is used. First an analogue to digital conversion (ADC) is carried out and the result, a 12-bit number, is the basis for the timer to generate time based interrupts. The specific code for the interrupts can be found in appendix F.

4.4.3 **Signal on change interrupt**

PortB of the microcontroller has some great features and one of these features is the ' interrupt on change'. When one pin of portB changes its value from 0V to 5V or from 5V to 0V, an interrupt is generated. These pins are connected to the external trigger input of the SPG. When the external trigger signal changes value, the SPG will activate a new

preset and write this to the audio matrix.

Take a look at figure 4.6. Since there are two different 'signal on change' interrupts, one for the rising edge and one for the falling edge it is possible to have the SPG step to the next preset on both edges, introducing modulation within the sequence when a pulse width modulation signal is connected. The preset will be activated in different time lengths.

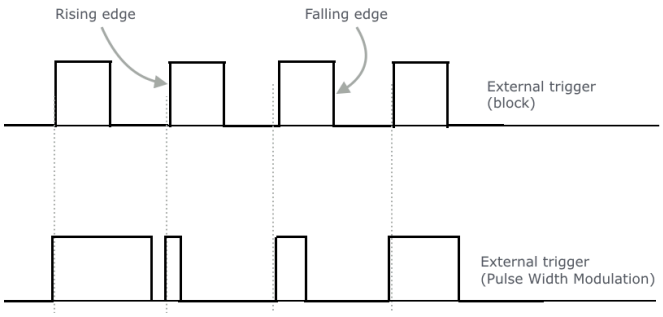


Figure 4.6. Example of positive- and negative edges with block and PWM.

4.5 **Sequence routine**

The second important routine defining the functionality of the SPG, is the sequence routine that steps through the local stored presets and is constantly checking the status of the received OSC-message (/cf string). The SPG can sequence based upon the internal timer, an external control-voltage and the external trigger signal. If the external trigger is active, which is checked within the initialization of the routine, it can switch on both edges or only on the rising edge(see page 38). The whole sequence flowchart is shown in figure 4.7. The upper part is the initialization of the sequence routine and the lower part is the actual sequence part.

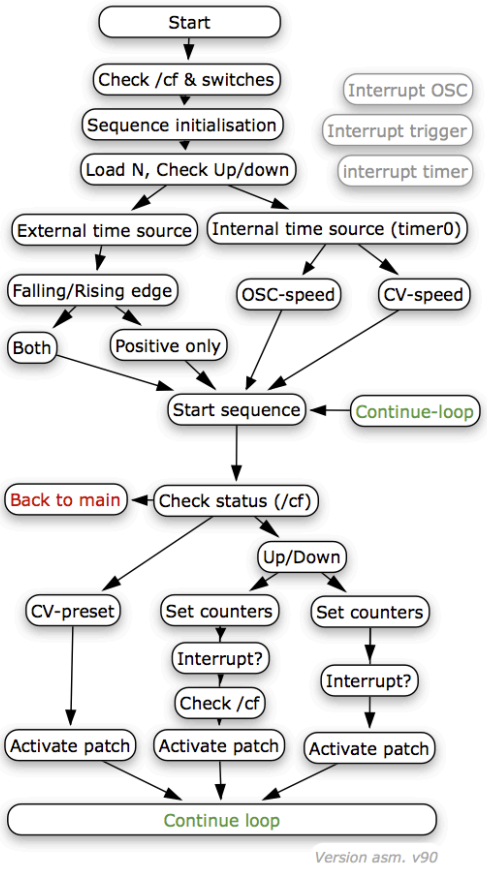


Figure 4.7. The sequence flow chart (version asm90)

As described in the chapter 'SPG Features' (page 35) the SPG can sequence up or down through the local stored presets or the order of the sequence can be determined by an external control voltage (cv-preset-id). If the order of the sequence is determined by the external control voltage, there is no timing involved. If the control-voltage changes its value, the preset linked to this value will be activated immediately.

The SPG design contains three sources that determine the timing or speed of the sequence: the received 16-bit OSC-number as part of the /cf OSC-message, the external control-voltage and the external trigger pulse. The number received by the /cf OSC-message is transferred directly to the onboard timer of the processor and activated. If the control-voltage to speed option is active, the AD-convertor measures the value of the voltage and the resulting 12-bit number is transferred to the same onboard timer. The external trigger does not work in combination with the onboard timer. The value change of the trigger pulse, generates an 'interrupt on change' and this controls the speed of the sequence.

4.6 **Max/Msp Patch**

To control the full functionality of the SPG, I created a software patch with Max/Msp,

that enables the user to fully control the SPG and generate presets in many different ways. It hosts the functionality to store many presets in special preset-banks and generate random series, or play complete pre-defined lists of presets. As will be stated in the chapter 5 'Features' (page 37), driving the SPG directly from the Max/Msp patch is very well possible, but it has its time or speed limits, which can reveal unstable behavior, especially at high frequencies. This is the reason why presets made within the Max/Msp patch can be uploaded to local memory of the SPG, so no new data has to be sent back and forth every time a preset changes; it saves time.

The red part within the layout of the Max/Msp patch is related to the uploading of presets to local memory and it shows the controls to set the SPG in the preferred configuration (figure 4.8).

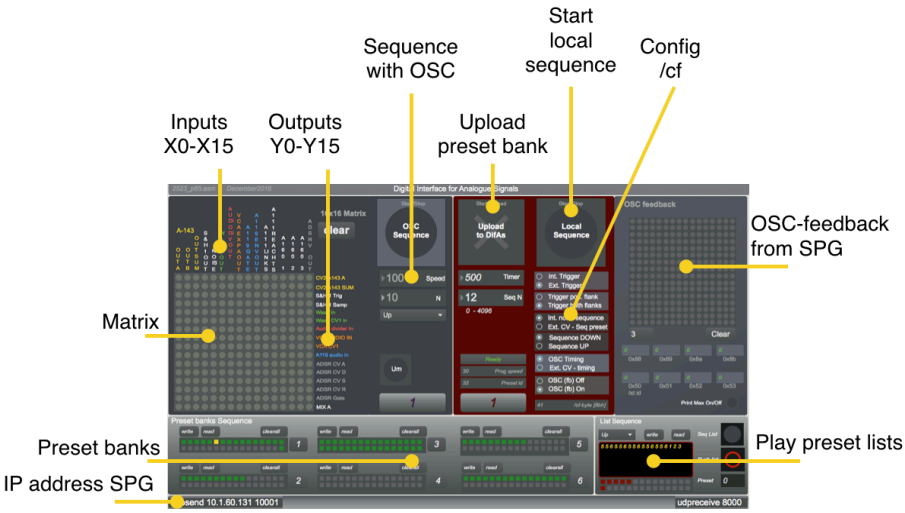


Figure 4.8 The Max/Msp patch, linked to asm version 85.

The configuration-bits, a part of the /cf OSC-message (see page 47) are the switches to set- or reset functions like external- or internal trigger and it enables the selection of different control voltages. On the left side of the layout the actual matrix is located with its inputs on top and the outputs on the right.

Below the actual matrix, presets can be stored in preset-banks. In this patch setup 6 preset-banks are created with each 32 presets. If a certain preset-bank is selected, indicated by the big number in the middle, it is this bank which serves as the source of the sequence. The order of the sequence is limited to up, down or pendulum and there is also an option to have the sequence-order determined by a list.

On the right from the matrix, the control to start/stop the OSC-sequence and change speed and preset-order.

On the right side of the layout another matrix is located. This matrix shows the received

OSC-messages from the SPG itself and provides the user with accurate feedback. The number-boxes underneath the feedback matrix show the content of the actual registers within the microcontroller. During the design and test-phase it's very practical to have this information available, but for the future user it is optional to use. A more detailed description on the Max/Msp patch can be found in Appendix G.

The presets are stored in the designated preset banks. In the right corner of the patch, the numbers can be edited to generate a list of preset-sequences. The core of the Max/Msp patch is based on the standard matrix-control object. This object generates row- and column data with the corresponding value of the cell. If a cell is selected, the output generates [column-number], [row-number], [0 if off and 1 if on]. This series of three numbers is converted into the right OSC-message pointing at the right cell within the 16x16 matrix.

To convert the standard matrix-control output to the right OSC-format, some complex steps have to be taken to achieve this conversion (see figure 4.9).

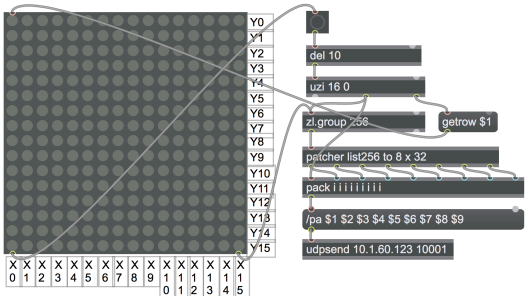


figure 4.9 Max/Msp standard matrix object with conversion.

5 **VC-SPG Features**

The main basic structure of the VC-SPG is described in chapter 2 and the more technical considerations are described in the chapters 3 and 4. In this section the focus will be on the features of the SPG. How does the sequence through presets actually work and what are the specifications of the external control signals? What about the sound quality of the SPG and are clicks audible when all 256 switches change position simultaneously?

5.1 **Initial startup**

Initially the SPG should be connected to the computer with a standard network-cable and it has to be connected to the power (-12V, +12V, +5V and GND). The Max/Msp patch that drives the VC-SPG generates the 3 main OSC-messages to drive and upload presets. Due to the large number of matrix options it's a good habit to keep a concise and clear labelling approach.

If all connections are made, the patching can start. Clicking on the crossing in the matrix-object will change the colour from gray to yellow, indicating the connection is now active. By selecting multiple cross-points and thus making multiple connections between inputs and outputs of different modules, sounds can be generated. If the preset, or patch sounds like it needs to be stored, the standard preset object located in the left down-corner of the Max/Msp patch can be used. A preset can be stored by Shift-Clicking a location. The small 'dot' will change colour from gray to green, indicating the preset is stored (see page 32).

5.2 **Audio in's and out's**

One of the main features of the interface is the input-adding function, which is an important part of the design. By selecting multiple inputs of the matrix to be routed to the same output, the real sum, or mix, of the signal will be available at the output. The summation of the signals is obtained with the use of Opamps[12] configured in a double inverting-amplifier setup. On the left side of figure 5.1 the input X0 is selected to be routed to output Y0 (the blue dot). Because both input and output of the matrix are buffered with an opamp circuit configured to be an inverting amplifier, connecting them together with the matrix will create a series connection of two independent inverting-amplifiers. A twice inverted signal equals a not inverted signal. The output result would be: -Y0 = -X0, or Y0=X0

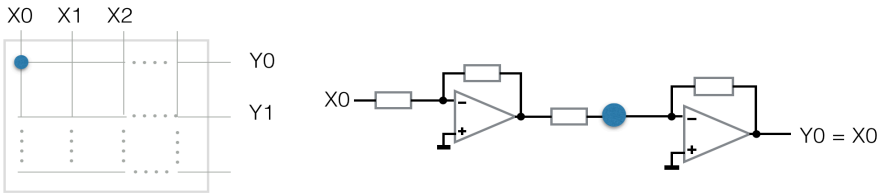


Figure 5.1. Opamp configuration with X0 routed to Y0 (blue dot).

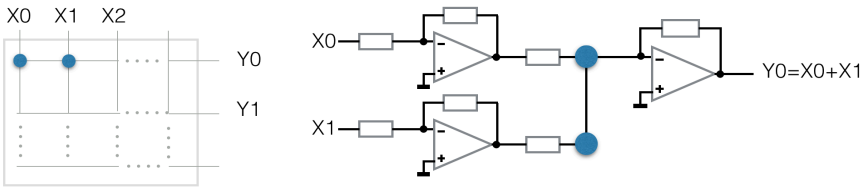


Figure 5.2. Opamp configuration with X0 and X1 routed to Y0(Y0=X1+X2).

If two inputs are routed to the same output a circuit configuration like figure 5.2 will appear. If n inputs are selected to be routed to one output it will always be a perfect summation of cv or audio signals. If multiple input signals can be routed to multiple outputs, the inputs signals also can be split.

When splitting one input-signal to be routed to multiple outputs, the amplitude of the original signal will decrease due to the opamp-design - improvement is still possible. In some applications this can be an interesting feature, but if the amplitude of the control-voltage drives the pitch of an oscillator, this signals a weak point of the design and something that has to be improved. The decrease of the amplitude when an input is split into multiple outputs is caused by the design of the opamp-circuits around the matrix itself (see figure 5.3). The moment an input signal is split, the two (or more) negative inputs of the operational amplifier are 'tied' to one point. The output of the opamp will decrease in amplitude.

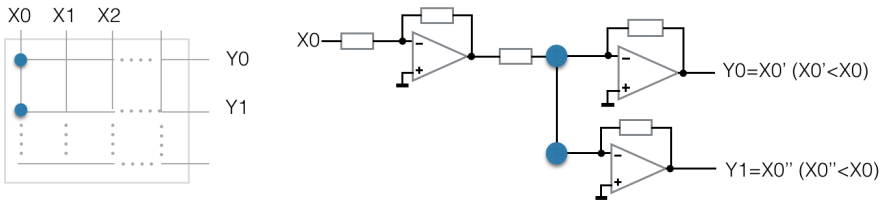


Figure 5.3. The split of one input to multiple outputs.

The printed circuit board (see appendix H is designed to also have access directly to the ins and outs of the AD75019 chip without the addition of the operational amplifiers. For some applications that do not require opamps in the circuit, this can prove to be a better solution.

5.3 **Sequence with the SPG**

What does sequencing mean for an interface that can store presets? Sequencing with the SPG means activating presets one after another, forward, backward or CV- dependent, with or without a controlled time-delay in between. This time-delay is determined by external processes and is variable. In sequence-mode the SPG can be driven and controlled by OSC-messages received from the host-computer or it can be controlled with multiple types of external analogue control signals. When it is directly driven from the computer by means of OSC-messages, complex series of sequences can be generated. Figure 5.4 illustrates the different sequence modes in which the SPG can operate and in the following section the different sequence modes will be discussed.

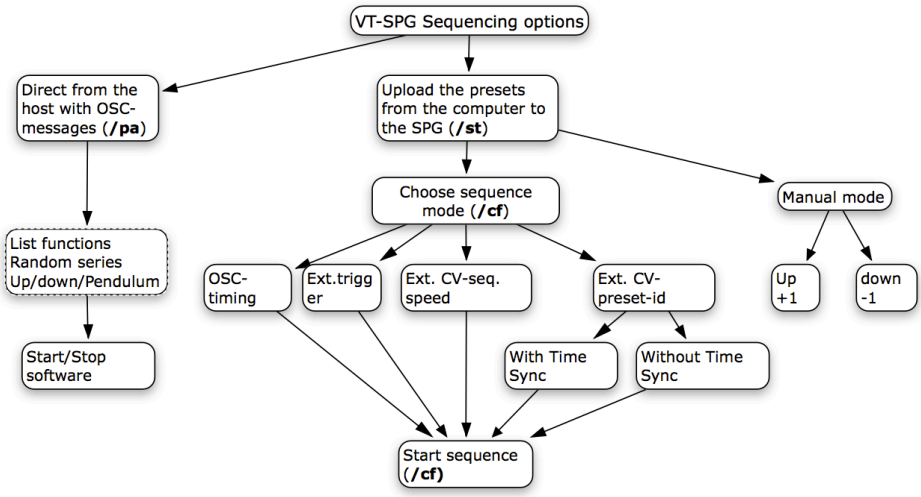


Figure 5.4. Different sequence modes: with computer, manual or ext. signals.

5.3.1 **OSC as sequence source**

The most complex series of sequencing through presets can be realized with the host-computer connected and driving the SPG directly. With software like Max/Msp or SuperCollider extensive series and patterns can be generated, but this approach also has a minor down-side. The maximum sequence speed limit is determined by the combination of software used, the network-speed and the receiving speed of the microcontroller in combination with the X-port.

The process of writing a set of 256 bits (= 256 switches simultaneously) from the processor to the switch array takes time. It is this time that determines the maximum speed at which the SPG can sequence through presets. It takes 568µsec (= 568 x 10^{-6} seconds) to write a whole string from the processor to the switch-array. This means the maximum frequency at which the SPG can sequence is 1.76kHz. The limitation of the speed is caused by the speed of the processor and not by the switch-array.

This maximum speed can't be achieved with the host-computer as the source, simply

because of the fact that the microcontroller simultaneously has to process the incoming OSC-messages and that process consumes time. This is also the main reason why sequencing through presets in local memory is much faster - the data does not have to be transferred anymore, which results in a considerable increase of sequence-speed.

In order to gain speed with the sequence and to reach these potentially interesting audio-rates, the best and fastest option is to sequence through the presets that are locally stored.

Stepping through these presets can be done with multiple settings (see figure 5.4). From the host-computer the sequence can be started/stopped by sending the **/cf** OSC-message (page 43). This OSC-message also contains a 16-bit number that is transferred into the local timer of the microcontroller.

When the sequence 'OSC-timing' starts, the time-base of this sequence is created by this internal timer which produces a very stable sequence. The internal microcontroller timer that is used to create this timing, has a resolution of 16-bits (=65536 values) which is extremely precise. If the speed of the sequence has to be adjusted during operation, this can be executed by sending a single /cf OSC-messages and store a new number into the internal timer. The sequence speed will change immediately. The maximum time t-max is around 2 seconds and the minimum time t-min is set to be 568 µsec (= minimum time the microcontroller needs to write all 256 bits).

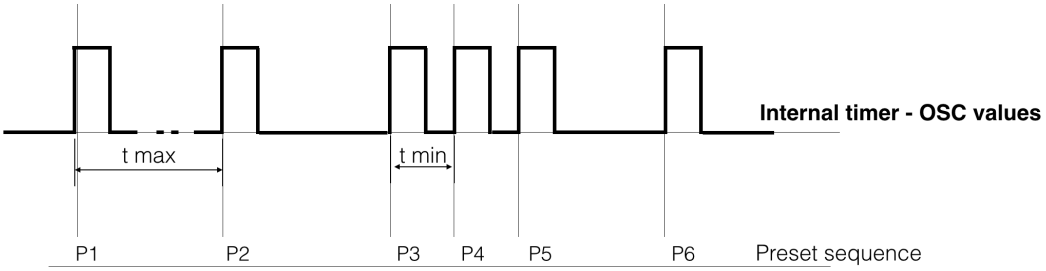


Figure 5.5. OSC-timing preset sequence.

5.3.2 **External triggers**

Instead of using the internal timer and the number received by OSC to step through the locally stored presets, the master-clock for the sequence can also be abstracted from an external pulse or trigger (see figure 5.4). This feature makes the SPG exceptionally interesting to be applied in an analogue setup because it can be part of the timing or sync-pulse of the studio itself. Another feature is the possibility to switch between two different trigger modes: switch on the rising edge only or switch on both the negative- and positive-edge of the external trigger. This will create duration differences between the sequential presets and it will double the sequencing speed, as shown in figure 5.6. When the SPG only steps to the next preset triggered by the rising edge (0V-5V), there's no pulse width

modulation (pwm) possible between the presets.

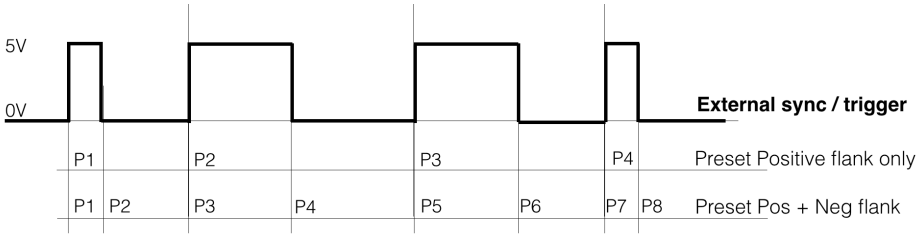


Figure 5.6. External trigger with preset order on only the rising edge and both rising and falling edge.

Notice the difference in timing. When the external trigger only switches on the rising edge, the time slices between the presets are regular - of course this depends on the symmetry of the used trigger signal. If the switching is triggered by both the positive and the falling edge, the time between the presets can be modulated by using Pulse Width Modulation (PWM).

5.3.3 **External Control voltages**

Besides the external trigger, the SPG can also be controlled and adjusted by two external control-voltages, both adjusting different parameters within the sequencing process. The CV-speed input changes the speed of the sequence that can vary between 568uS (absolute maximum speed) and around 2 seconds. The other control voltage input is linked to the preset-id.

5.3.3 **Control-voltage to speed**

The internal analog-digital convertor (ADC) of the SPG has a 12-bit resolution. Incoming voltages between 0V and 5V are converted into numbers between 0 and 4095. These numbers are used to be the source of the timer that controls the sequence speed. When the control-voltage slowly increases from 0V to 5V, the sequence speed will increase at the same rate - see figure 5.7.

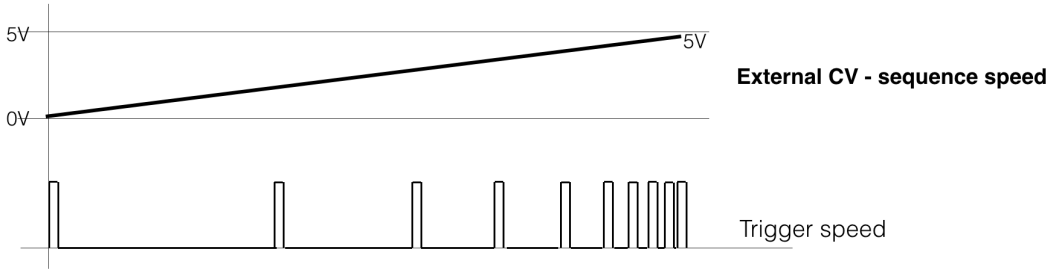


Figure 5.7. Control voltage to sequence speed.

When the SPG CV-speed input is connected to an analogue signal, the speed of the sequence through presets will be modulated accordingly. The maximum speed that can be reached is 1,7kHz, which corresponds with the maximum input voltage (5V) at the CV-input.

5.3.4 **Control-voltage to preset-id**

To introduce a less predictable and more musically interesting sequence-character (not only sequencing up and down) the possibility of converting a control-voltage to preset-id is introduced. When the cv-speed changes value, also the active preset-id will change value.

The AD-convertor has a resolution of 12-bit and this results into a number between 0 and 4096. These numbers have to be re-mapped to preset-id numbers that can vary between 1 and maximum 32 (only 32 presets can be stored in local memory).

To make this feature feasible and musically interesting, a way of re-mapping the resolution is implemented. If for example only 5 presets are stored in local memory and the cv-preset-id results into numbers between 1 and 32, chances are, it will never reach or activate the right preset. In order to avoid this issue a choice can be made concerning the resolution of the control-voltage. When the SPG is programmed and controlled with the / cf OSC-message, a number N can be sent and stored (see Max/Msp patch explanation). This number, first of all, defines the amount of presets within the sequence and therefore sets the maximum, but it also sets the range of the cv-preset - the resolution. Please refer to figure 5.8.

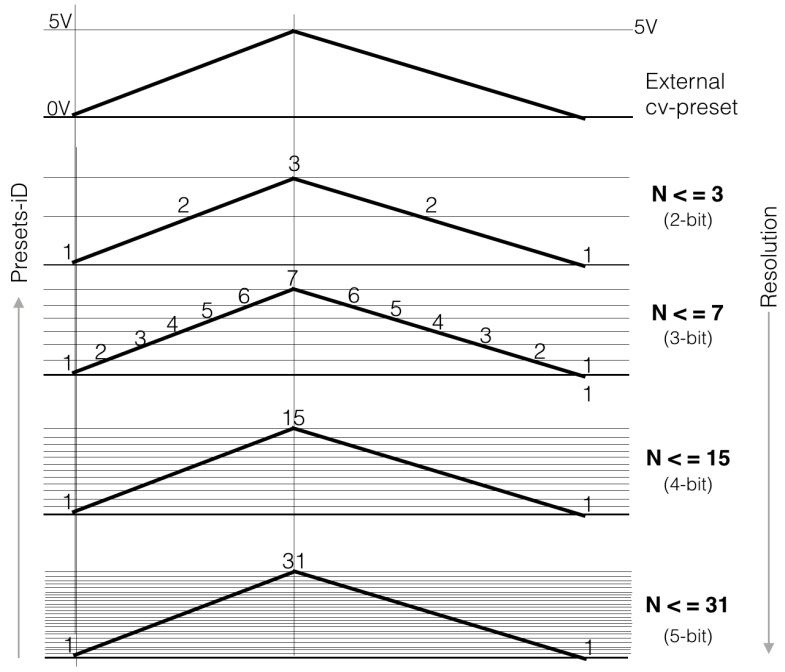


Figure 5.8. Representation of the 5 different resolutions used (N=1,3,7,15,31).

If the number N = < 3, the control voltage (0V-5V) will solely be mapped to these 3 presets - a resolution of 2-bit. An increasing value for N, results in a higher resolution, with a maximum of 32.

5.4 **Audible clicks**

If an audio-signal on one of the inputs is routed to an output, and the output itself is connected to a speaker or another audio device, the sudden switch from one input signal to another input signal can cause 'clicks', or unwanted pulses (discontinuities). This phenomenon is caused by the abrupt change of the value the audio-signal has at the moment of the switch. Switching audio-signals without an audible click is only possible if the moment of switching is in-sync with the zero-crossing of the signal or when the two input signals have exactly the same amplitude and slope (speed of increase/decrease).

Furthermore, this aspect constitutes an integral sonic aspect of the system. As such the design considers this immediate switching as a distinctive feature rather than drawback of the system.

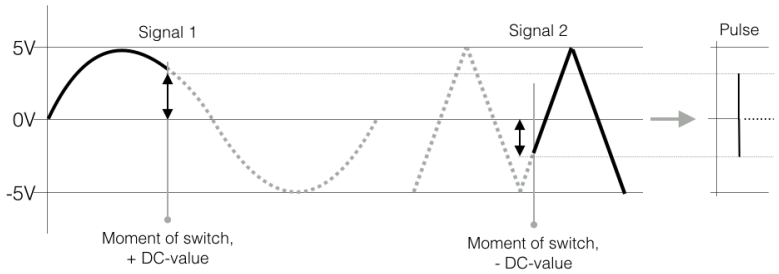


Figure 5.9. Example of a switch from signal 1 to signal 2.

As shown in figure 5.9, if signal 1 is suddenly changed to signal 2, the amplitude at that moment will change from positive to negative, introducing a discontinuity in the signal - in the example it actually will move the speaker backwards, resulting in an audible click or pulse.

The clicks can be avoided if the audio-signals and the routing within a preset are not switched but left untouched. Switching control voltages, driving VCO's or other modules with a CV-input, will (in most cases) not result in clicks. Another option would be to introduce gate's or a system with a fast fade-out and fade-in - all of these solutions will also alter the audible result and maybe even create interesting material.

5.5 **SPG Switching quality**

To investigate the quality of the switch-array in combination with the firmware driving the switching-process, a test with a spectrum-analyzer[13] was carried out. By connecting +5V to input X0 and sequence between two presets that connect and disconnect X0 to Y0

(so only one connection is switching ON and OFF), a regular square wave is generated on output Y0. By using a spectrum-analyzer connected to output Y0, the harmonics of the square-wave can be detected, which are an indication of the quality of the signal. A perfect rectangular wave-shape consists only of odd-harmonics - a summation of the fundamental frequency and the odd harmonics (3,5,7,9,…). If there is some imperfection or irregularity present in the signal, also even harmonics will be visible, indicating a lesser perfect signal.

In our measurement with a spectrum analyzer, only odd-harmonics were visible at lower and higher frequencies. This measurement indicates that the SPG has a very precise timing, which is an important feature considering it is used as a compositional tool or musical instrument.

6 SPG communication

Within the SPG the main communication is executed through the OSC-messages between computer and VC-SPG. OSC is using the UDP protocol as a carrier throughout the internet but also a robust and well known protocol like RS-232 for the data transfers between the processor and the X-port. The internal IC's, or Integrated Circuits, communicate by means of Serial Peripheral Interface (SPI) to transfer data back and forth. In this chapter the focus will be put on the different ways of communication and the tools and various protocols that are used within the VC-SPG - see figure 6.1.

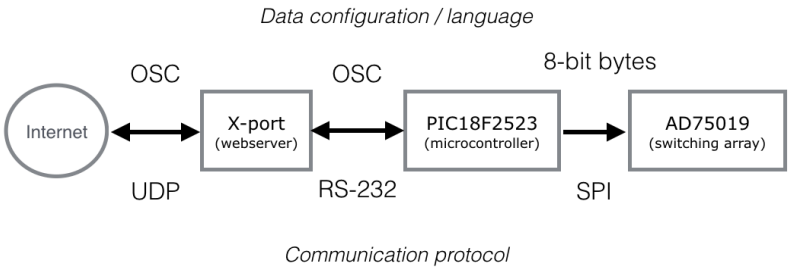


Figure 6.1. All used data protocols in the SPG.

6.1 OpenSoundControl (OSC)

Open Sound Control (OSC) is a protocol for communication among computers, sound synthesizers, and other multimedia devices that are optimized for modern networking technology. Within the design of the SPG only OSC-messages are used to communicate with the external computer. An OSC-message consists of three parts: an OSC address-tag, an OSC type-tag and the OSC argument(s). Figure 6.2 shows the generic OSC-message that is used for driving the matrix.

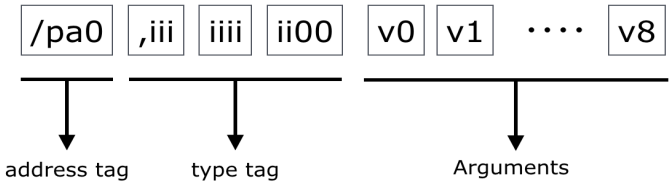


Figure 6.2 SPG OSC-message configuration

All OSC-communication is achieved with blocks of 4 bytes (1 byte = 8-bits) and the characters use the standard ASCII coding[14] for the definition of the right symbol. In figure 2 you will notice the extra 'zero' after the address tag '/pa0'. This zero acts as a separator between the address-tag and the type-tag and it completes the complete data-string to a multiple of 4-bytes. After the definition of the type-tag consisting of nine 16-bit integers (9 x i), two zero's are added again to complete the multiple of 4 and to accomplish the separation between the type-tag and the OSC-arguments. Both the /pa and the /st strings

contain a total of 9 integers [v0 - v8]. For the communication with the SPG, the address-tag (/pa in figure2) can have three different values:

/pa : this address-tag is used if the SPG has to activate the new received value instantly. 'Pa' is an abbreviation for 'Patch'.

/st : for the storage of a preset the OSC-message should begin with /st (storage).

/cf : this short OSC-message, consisting only of one OSC-argument is used to configure and control the SPG.

6.2 **Writing to the audio matrix**

To be able to write different matrix presets to the audio-matrix, it is necessary to write all 256 bits every time the presets changes in order to obtain stable behavior. This corresponds to the setup and the design philosophy of the Analog Devices AD75019 matrix chip. The Analog Devices AD75019 has 256 switches onboard that all need a specific value before the new status can be activated, so all 256 bits have to be transferred every time a new preset is created.

Figure 6.3 shows the build-up of the content of the arguments used to transfer presets from the host-computer to the SPG or vice versa. The first byte of the OSC-message, byte1, is a 32-bit integer and it holds the matrix setting for two rows: X0/Y0 to X15/Y1. The next byte (byte2) holds the next two rows: X0/Y2 to X15/Y3. Notice the 'Yx' is increasing and that every byte always contains twice the settings from X0-X15. The first byte of the OSC-message, byte0, contains the preset-id and has still multiple bits assigned for later use.

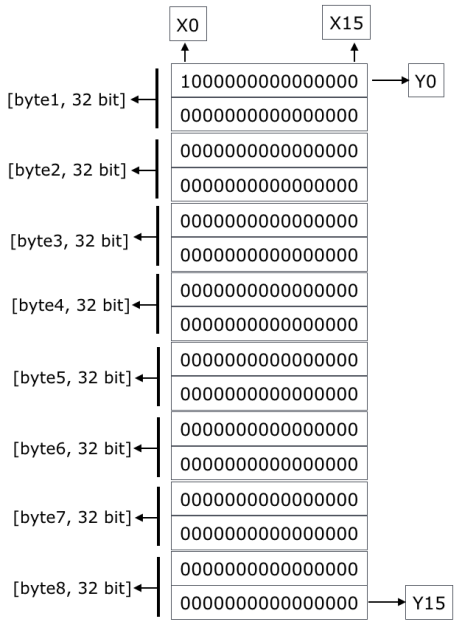


Figure 6.3. Bit representation of the matrix

6.2.1 **The patch OSC-message (/pa)**

/pa [byte 0] [byte 1] … [byte 8]

The OSC-message starting with '/pa' (patch) is directly activated within the SPG when it is received. It will instantly switch On/Off the corresponding switches in the matrix and activate the preset. The first byte [byte0] is an extra byte which does not contain switch information, but it can hold information to configure the SPG. Since assembly version v85, [byte0] is empty for the /pa string, it offers place for future data to be stored - [byte1] to [byte8] represent the 256 switches in the matrix. Check figure 3.

Example 1

If you want to switch ON the X0 and Y0, you have to set the first bit of [byte1] to 1. This is switch X0/Y0. The last bit of [byte1] represents X15/Y1. See figure 4

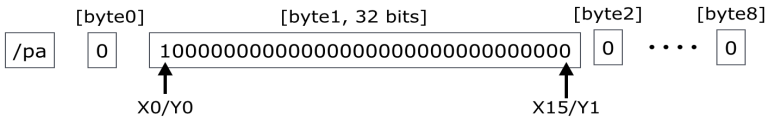


Figure 6.4 Switch ON X0/Y0

Example 2

Switching on only Y15/X15 (this is the last bit in the row of in total 256 bits):

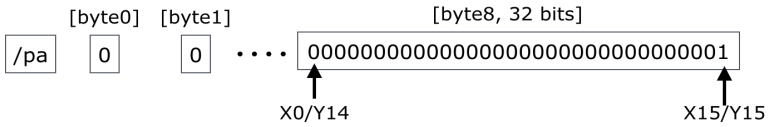


Figure 6.5 Switch ON X15/Y15

Example 3

If you want to switch OFF all the connections of the matrix, you have to send all zero's and all the matrix connections will be closed: /pa 0 0 0 0 0 0 0 0

6.2.2 **Store OSC-message (/st)**

/st [byte 0] [byte 1] … [byte 8]

If the presets send to the SPG have to be stored in local-memory for later use without the computer being connected, the OSC-message should start with the address-tag '/st' (store). If the SPG receives the OSC-message with this address-tag, the preset attached will be stored in local-memory at the location indicated by the preset-id in [byte0]. The first byte [byte0] also contains data that determines the amount (N) of presets that will be part of the sequence. The remaining two bytes do not have a purpose (yet) and can be used for future development ideas

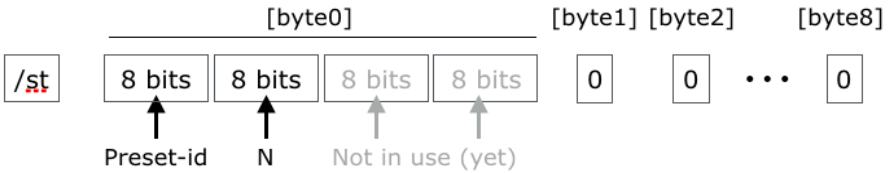


Figure 6.6 Storage OSC-message

Like the /pa message, [byte1] to [byte8] contain matrix switch information - see figure 6.4.

6.2.3 **Configuration OSC-message (/cf)**

/cf [byte0]

The '/cf' or configuration is a single byte string and contains data to change settings or configure the mode of the SPG. The fist versions and prototypes were supported by physical external switches to start/stop the sequence, or to make a choice between different timer sources. Since assembly version v85, the switches can still be used, but the configuration byte can 'set' or 'reset' the same functions. It can actually override the switch settings.

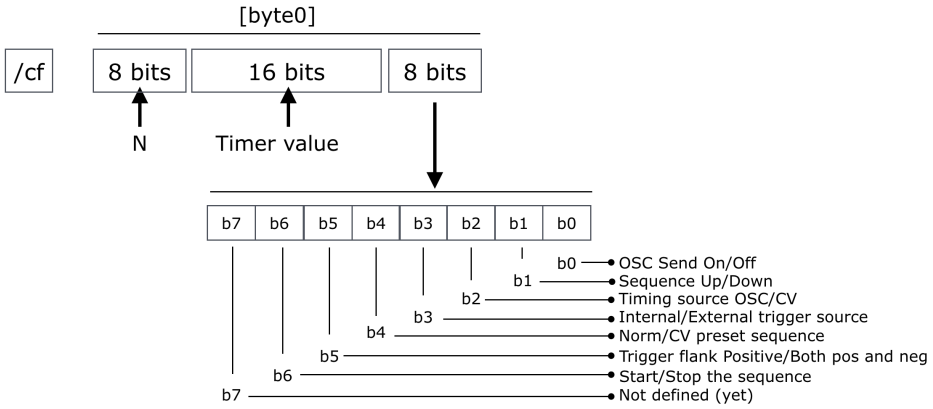


Figure 6.7 Configuration OSC-message

Figure 7 shows the setup of the one data byte [byte0] which defines the configuration. The 8-bits below b7 - b0, also called the LSB bits or Least Significant Bits, contain the switch values. The switch values are explained in more detail in the quick reference. The 16-bits in the middle contain the timer value. This value determines the sequence speed when the timing source is set to OSC. The 8-bits on the left (MSB or Most Significant Bits) contain the value N and it sets the amount of presets of a sequence.

6.3 **SPG and network communication**

The communication between the microcontroller inside the SPG and the internet is controlled by the embedded web server, called the Xport (see page 14 for more details about this component). The protocol (or carrier) that is used to transfer the OSC-data from the sender to the receiver is called UDP (User Datagram Protocol).

The UDP protocol is one of the most used protocols on the internet, because it is specially designed to transfer data at high speeds. Therefore it is mainly used for video-conferencing, phones and online gaming - applications where a fast reaction-time is important. With UDP there is no control mechanism that will check if all the data has arrived, because the loss of one package does not really influence the process that much. The TCP/IP protocol, mainly used for communication between websites, checks if every package has arrived in order to show a complete website and indicate an error otherwise.

6.4 **SPI serial protocol**

The language that is used to communicate between devices or peripherals within electronics circuits is called SPI , or Serial Peripheral Interface. The communication between the audio-matrix switch array and the microcontroller is SPI based. By means of three different signals, the data is transferred from the source (the microcontroller) to its destination (the switching array). The first signal is the Clock. A clock signal provides the timing and the trigger to start and 'do' an action. The second signal is the data line. Every time the clock transitions from low to high, the value of the data line is being 'clocked in'. If all data-bits have been clocked in, the third signal, the Shift Clock, will activate the new received value.

In the chapter Software design, page 24 an example is given how the SPI protocol is used to transfer an 8-bit byte from the microcontroller to the audio switching matrix.

6.5 **RS-232 serial protocol**

RS-232 is an old communication protocol, dating from the time computers were not yet part of our daily life and, at this point, is still in use today. RS-232 was introduced in 1960 by the Electronic Industries Association as a Recommended Standard and being revised over the years, it is still one of the most common communications protocols used. The data-transfers between the microcontroller and the Xport within the SPG uses the RS-232 protocol. Even today all modern computer boards (Arduino, RaspBerryPie, BeagleBone, Teensy etc), use RS-232 to communicate.

7 Practical applications and roles

Within this section the focus will be on the practical applications of the SPG and the roles that it plays in these applications. Since the design and production of the SPG interface, it has become more and more clear that the different roles the interface plays and the applications for which it is used are more diverse than I initially thought. To be able to drive 256 switches at once at high or low speeds, driven and triggered by external signals, offers many possibilities for a variety of musical and artistic applications.

The SPG has already been used by several students and professionals during this research and they provided me with very useful feedback of which some is already implemented in the current design. In the first week of November 2016 I initiated a 'SPG-workshop', that enabled different students to practically work with the interface and share experiences. During this workshop we discussed the diversity of the possible application fields and practical applications, shown below in figure *7*.1.

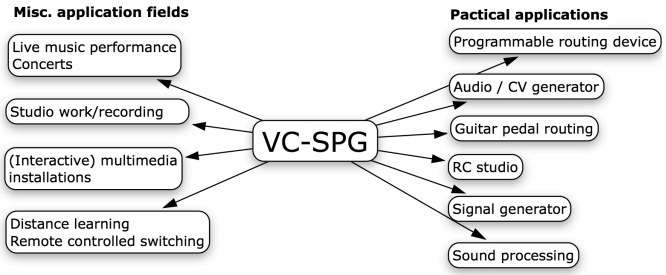


Figure *7*.1. SPG application diagram.

7.1 VC-SPG roles

The VC-SPG offers the possibility to change the position of 256 switches in a matrix of 16 inputs by 16 outputs at once at high speeds. As we will see it is a very powerful instrument for many applications. The role of the VC-SPG within these applications, could be divided in a few categories.

The first role of the SPG is the functionality of 'a programmable routing device' with memory locations: changing the routing of signals with the computer and create unique signal-paths for different situations or applications is the most common feature that is used in the practical examples.

The second role of the SPG is the one of a generator. Since the SPG can switch so fast that the resulting signal enters the audio-domain, it becomes a generator that is capable of generating and modulating audio and control voltages. And as generator and modulator, the application field of the SPG expands.

The third role is the remote signal routing possibility. Since the SPG uses the ethernet cable and UDP protocol for its control it can be directly operated over a local network or

the internet. Therefore the SPG can be used to control processes, which are on the other side of the globe, or next door - purely by using the internet infrastructure. It therefore can be an effective tool for remote controlled instruments, studios or interactive installations. It can even add more functionality for Distance Learning (DL).

Application \ *Role or function*	Programmable routing device	Audio / CV generator	Remote signal routing
Signal Routing (modular synth)	●	●	●
Sequencing (modular synth)	●	●	●
Live performance (electronics)	●	●	●
Audio-mixing (audio summation)	●		●
Guitar pedal automation	●		
Interactive installations	●		●
Switching electronic components	●	●	●
Sound processing	●		
Distance learning projects	●		●

Figure 7.2 Table of applications and roles.

7.2 The programmable routing device

The first application and role is the one of the 'programmable routing device' for physical connections or signal-paths - this was the initial idea and motivation to develop the SPG. It can store up to 32 presets (32 x 256 switch positions) in local memory. A typical application is the analogue studio: the physical connections can be stored and re-called when needed. Switching the studio-setup to be ready for a next job or new soundscape becomes as simple as the push of one button.

At this moment one SPG-model is designed for the use in euro-rack modular synthesizer systems and has one physical switch which activates or de-activates the manual-mode. When the manual-mode is selected , the 'up' and 'down' push-buttons can be used to recall the desired preset. Visual feedback is provided with a simple setup of 5 led's showing the 5-bit representation of the active preset-id. Figure 7.3 shows the SPG-model that is embedded in a Doepfer A-100 system.

The amount of memory available in the SPG is not that big, only 32 locations, but so far this seems to be more than enough for the applications that it has been used in. In case the 32 memory locations are not enough the memory can be updated very fast (in about 20mS) by uploading 32 new presets from the computer with OSC-messages. In future developments of the SPG an expansion of the memory, by embedding an extra memory chip, is not that complex and is absolutely feasible.

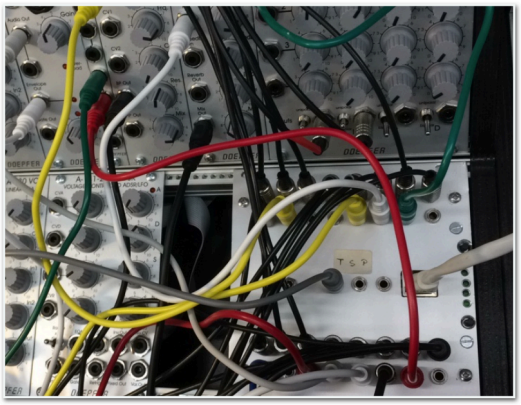


Figure 7.3. SPG model embedded in Doepfer A-100 system

7.2.1 Signal routing and automation

An obvious application of the SPG is the one of signal-routing. Looking at the SPG from a more musical point of view, switching the routing of multiple guitar pedals can be a really interesting application. Guitar players generally use a lot of effect pedals to condition, modify and tweak their guitar-sound to be a perfect sound for that particular song. In normal use, these guitar pedals are all connected in series and in a certain static order. The output of the first pedal is connected to the input of the second one, so creating

a serial connection.

Of course there are also combined effect pedals, into which the user can program the combination sounds as well as the complete routing and save it in a preset. The downside of these digital-pedals (in general) is the sound quality. The sound-quality that is produced by these digital multi-effect processors is somehow 'thin' and very digital - at least that's the opinion of most guitar players.

If good quality pedals (mostly rich sounding analog effects) can be combined and re-connected through a matrix like the SPG, sounds can be programmed and re-called without the loss of the good audio quality. A multi-effect pedal can be created from these special pedals the guitar player already has. Even feedback loops can be part of the routing, introducing new areas of sounds in many ways. It is also possible to change the order of the pedals, which really influences the sound.

Since the SPG also has an external trigger input, the sounds of the guitar can even be changed by some external sequence for example, creating interesting musical possibilities, especially for the guitar player.

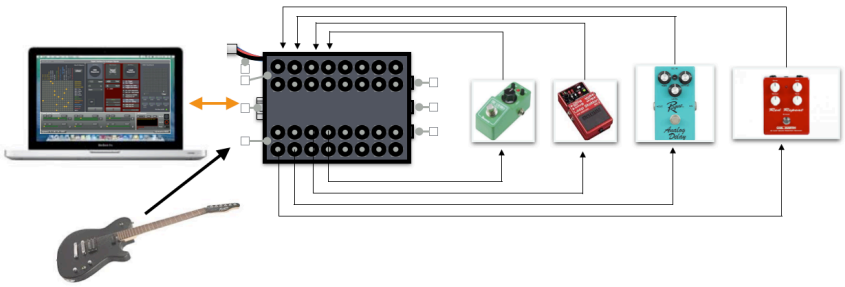


Figure 7.4. Example of the guitar pedal setup.

7.2.2 **Live performance tool**

In the situation of a live performance, the VT-SPG can be a practical tool to change presets between or during different pieces, e.g. the guitar pedal setup. When a musician plays a modular-synth for example, re-patching the cords during a song is something that will not happen because the risk of making an error is too high. When the patch-cords are routed through the SPG, it is easy to jump to the next composition and change cables by selecting a next preset.

Another interesting application would be to introduce the element of surprise within an electro acoustic ensemble while playing live. Imagine multiple musicians all having a SPG embedded in their own setup and influencing each others routing. This re-routing of signals can be part of a game-performance setup as well.

7.2.3 **Remote Controlled Studio (RC-studio)**

The RC-studio, already referred to in the pre-research section (see page 6), is also a practical application of the SPG and actually the original starting point of this research project. Within the RC-studio all audio and cv signals can be routed to all modules within the studio setup. Feedback loops can be made and complete lists of presets can be played back.

7.3 **Audio and CV generator**

One of the first devices added to the test-setup of the project and what revealed to be a powerful application of the SPG was a box consisting of 10 potentiometers (figure 7.5). All of these 10 pot's are connected between +5V and GND and are thus able to generate 10 different DC control-voltages between 0 and 5V. If these potentiometer outputs are connected to the first 10 inputs (X0-X9) of the SPG and the uploaded presets will sequentially connect one input to output Y0, a wave-shape will occur shown in figure 7.5. The frequency of the wave-shape depends on the speed of the sequence.

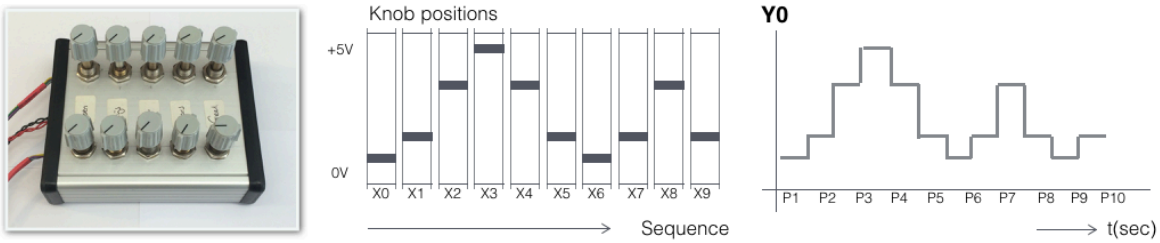


Figure 7.5. 10-channel potentiometer-box and the possible resulting wave-shape on Y0

Due to the summing-opamps, which are added to the inputs and the outputs of the SPG (see page 35), multiple signals routed to one output will be summed. If for example multiple inputs have dc-values, the output is a perfect summation of these dc-values. This application makes the SPG an interesting generator for driving miscellaneous modules in any analogue studio.

The generator role or function is also interesting if the connected signals all have different frequencies or AC-values. The summation signal that occurs on the different outputs of the SPG are modulated in the sequence speed of the generator, resulting in complex and interesting sounding audio-signals.

Another interesting application is the use of the 'control voltage to preset-id' input being connected to one of the SPG outputs. As such, making a loop, nice unpredictable generator patterns can be the result, especially if the SPG is part of a feedback system in an analogue studio setup.

7.4 **Remote signal routing**

A relative new phenomenon, which is getting more popular in the world of education, is the possibility to use the internet to have teachers teach classes over long distances - you can call it "super-skype". A group of students in New York (for example), can follow lessons given by a teacher somewhere in Europe or vice versa. The fast internet connections make this option feasible and the quality and speed of the connections are continuously improving. The Royal Conservatoire in Den Haag is using amongst other software packages, the LoLa (Low Latency) system[15] which enables musicians play together with the aid of the internet.

At this point there are many different applications to consider for the SPG. Since the SPG uses the internet as its main communication channel, sending and receiving OSC-messages by means of UDP, it could well be executed that during performance or distance-playing the performer can start or stop certain processes on the other side of the line. If the SPG is operating in combination with the Ipson CV to OSC- and OSC to CV-boards[16], users on both ends can be of great influence to each others artistic process.

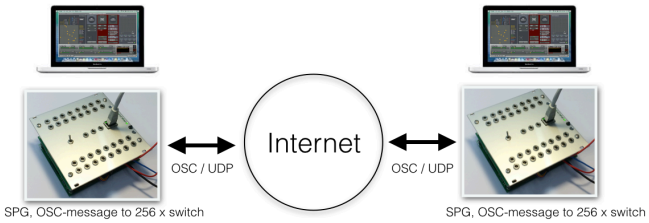


Figure 7.6 Possible distance learning setup

8 **Musical and artistic results**

Although I am an experienced designer of sensor based electronic-instruments, this particular research-project resulted in more than just a new musical-interface. It provided me with better insides of electronic music in general and the philosophy behind it. I've grown to be increasingly inspired by the enormous amount of sound creation possibilities. And I now understand the addictive force modular synthesizers have, when exploring the modular world.

Furthermore, I realize that the VC-SPG can be a great instrument for all kind of disciplines. The results for this project at this point can therefore be divided in two different sections: my personal experiences and experiments and the musical and artistic results created by different users during the last year.

8.1 **My personal results**

To be a student within the institute where I have been working at for many years, revealed new aspects and insights in my line of work. The weekly master-circle gatherings for Sonology, where all students (including me) had to share their work-progress, was an inspiring experience. It seems that when you are working on a project like a master research project, suddenly the work becomes more important, simply because you have to document the whole research and present it to an audience. However, that particular part of my work did not really change - I have developed a lot of new technology already and I documented this as good as possible.

The lessons I followed in the Sonology analogue studio Bea-5 broadened my view on the studio itself and gave me deeper insight in the importance of all separate modules installed in that studio. I used to be the technician working behind the racks, soldering and repairing electronics, now it was explained to me how to apply this technology in order create electronic music.

One of the first lessons, creating a pattern of tones with random pitches, is the one setup I applied in my experiments with the Doepfer A-100 system, where one SPG is embedded in the modular synth (see figure 8.1).

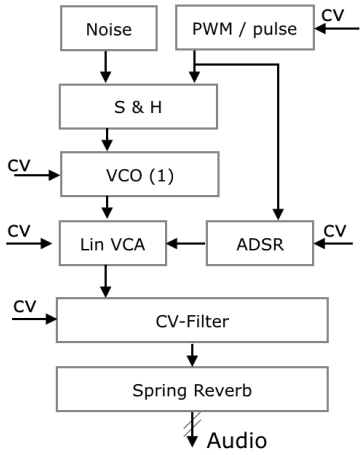


Figure 8.1 The S&H patch creating random pitched tones.

The noise as input for the Sample and Hold module generates random DC-values every time the S&H is triggered by the connected pulse. These random DC-values, connected to the frequency input of a Voltage Controlled Oscillator (VCO) produce tones with random pitches. In combination with an ADSR (Attack, Sustain, Decay and Release) envelope generator and a Voltage Controlled Amplifier (VCA), the amplitude curve of these tones can be modified (see page 53 for an explanation). If these tones also run through a voltage controlled filter (VCF) and a spring reverb, the audio results are, for me at least, very impressive.

The fundamental 'Sample & Hold' setup shown in figure 8.1 is reproduced with the SPG in a Doepfer A-100 system and fully controlled by the Max/Msp patch - see figure 8.2. All of the control-voltages are sequentially rerouted by the SPG and connected to different signal sources or VCO's. This creates constant changing control voltages, which results into a new layer of sounds. Of course all the existing knobs, can still be manually controlled and played with. Some audio samples of this experiments can be listened to at the links below. In the second sound example the 'control voltage to sequence speed' can clearly be heard.

1. https://soundcloud.com/user-520335425/a100_exp1
2. https://soundcloud.com/user-520335425/a100-exp2

Figure 8.2. On the left the Doepfer A-100 system with the VT-SPG build in and on the right, the actual Max/Msp patch.

8.2 **Artistic results**

As mentioned before I have organized a workshop of 4 days in which I have invited students to explore, challenge and use the SPG in their own proposed way. Furthermore I have asked some teachers to use and reflect upon the SPG. The SPG has been used is different ways and played different roles in their music production. The students and teachers shared their findings and experiences in diverse forms. I have received music, code, video and writing. In this section I will share some of the results.

Ruben Brovida (Sonology Ba Alumni, graduated in 2016)

The first musician that shared his experience with the SPG was Ruben Brovida, a graduated Sonology student. He really wanted to join the workshop and provided me with very useful comments and interesting audio-examples. Ruben is the owner of a modular synthesizer setup and he was really eager to start experimenting with the SPG.

Ruben: "The use of the SPG allows a different workflow both in the studio and in the live performances. The interface stores patches as drawings do, but you can very quickly recall them, allowing the user to keep developing and building up his/her own collection of patches, evolving them and saving them.

Aside from the recall features, the sequencing possibilities offered by the matrix, are interesting composition-tools, pointing at different directions in reference to the traditional patching. Working with a digital representation of the matrix also gives a different perspective on the possibilities offered by the modular system itself, showing new possible relationships between audio and control-voltages. Because of the hybrid approach, live performances (using modular synthesizers) can be prepared in a completely different way. Few modular artists change their patch on stage and for many different reasons usually sticking to only one pre-prepared patch for each set. The SPG allows to store 32 patches, so that user can even on stage quickly swap them and push the set in much less static directions."

Ruben has revealed multiple functions for the SPG within his experiments. Recalling and storing presets, using the switching of the SPG as a compositional tool and he noticed that the SPG shows less separation between audio and control signals. Another

observation Ruben made is that by connecting the ins and outs of the modules to the SPG, the synthesizer has a different graphical representation of the setup, which allows a different workflow and possible new compositional ideas.

Some sound samples Ruben shared:

https://soundcloud.com/user-520335425/feedbackpatch-ruben-brovida

https://soundcloud.com/user-520335425/matrixmixerlexday1-ruben-brovida

Max van der Wal (Sonology student Ba 3rd year)

Max shared some musical results of his experiments in analogue studio Bea-5 and he also documented his experience and workflow using the SPG.

Max van de Wal: "The first thing that I noticed was that the workflow was very different. It takes longer to set up but once that has been taken care of there is more freedom. I made several patches that I would have never made simply because of the fact that it only took a push of a button, therefore less thinking in advance was required and more time was left for experimentation. This possibly was more useful to me than the actual sequencing through the patches.

The sequencing I did by receiving a trigger straight out of the sound that was put out. Every time the voltage crossed a certain threshold an impulse was created. This impulse went into a variable scaler. This is a device which takes in a certain amount of triggers and outputs a trigger when the chosen threshold is received. Due to this, I could more or less decide how fast the changes would be, but it would always be different because the decision making would depend completely on the sound itself. This trigger would go into a sample and hold device that would give a constant output. This output would go into the preset-input in the matrix and thus make a new preset active accordingly. It took me quite a while to get this to work perfectly to my liking because of the way the voltage was scaled in the analogue studio. In the end, I made it work by doubling the voltage of what would go into the sample and hold. "MatriLex 4" and "MatriLex 5" are two phrases that are taken from the moment that the whole system worked perfectly".

Max used the audio result of the SPG for the 'control-voltage to preset-id input', creating feedback in his setup, which generated interesting results. He shared some short recordings.

https://soundcloud.com/user-520335425/matrilex-4

https://soundcloud.com/user-520335425/matrilex-5

Sabina Ahn (Artscience 2nd year master)

An art installation and, from a technical point of view, more practical approach of the use of the SPG is done by Sabina Ahn. The SPG is used in her art-work to route different source signals to different speakers — she uses the SPG as signal-router. Her graduation work (she will be the first one to use the SPG for a graduation project) consists of 10 big

columns filled with mud full of living organisms. Every column has an oscillator, a speaker and a bright led-light linked to it. These columns generate the energy for her electronic analogue oscillators and the SPG is applied to change the routing and create variations in the direction and intensity of the generated audio.

Sabina is traveling around the world with her work and the VC-SPG she uses travels along with her and she definitely wants to own one. She shared a series of beautiful photos and a link to a video of her work, titled 'Sonomatter' — see some impressions in figure 9.3.

Video link: https://vimeo.com/199717824

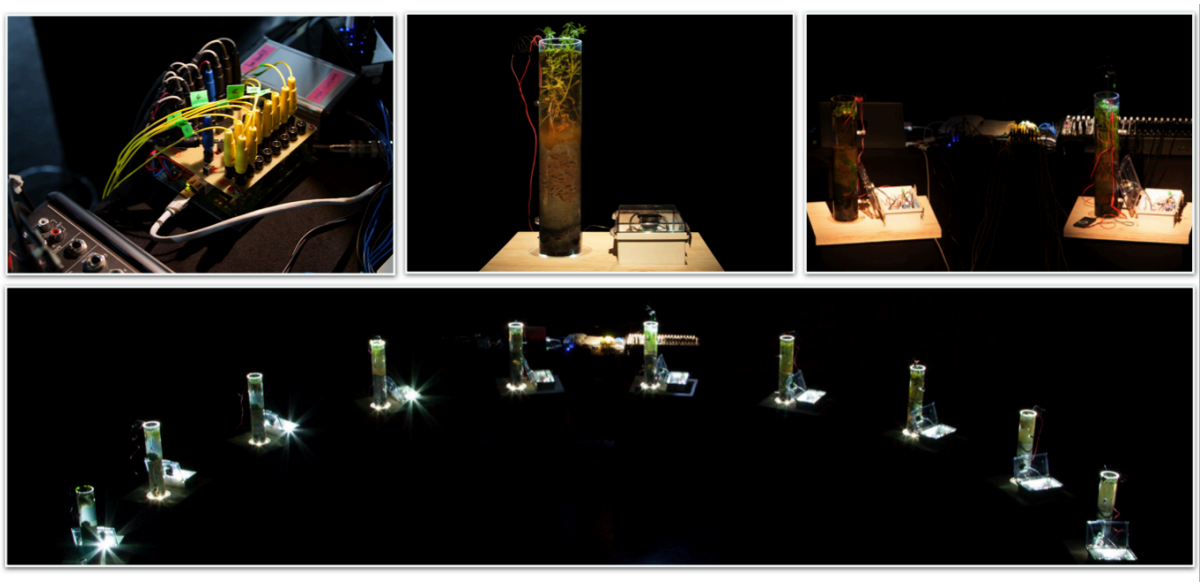


Figure 8.3 Impression of the setup of Sabine Ahn. In the left upper corner the SPG used for routing the different signals.

Justin Bennet (Sonology teaching staff and Alumni)

One of the first SPG testers was Justin Bennet. Justin is a member of the Sonology teaching staff and as an audio-visual artist he is always interested in new developments. He is also an owner of an EMS Synthi synthesizer and he was really curious what the effect of the SPG would be when the two were combined. He shared some findings:

Justin Bennett: "Firstly I adapted the Max patch provided by Lex so that I could prepare patches that were visually identical to an analogue EMS patch. The EMS matrix has inputs on the vertical Y axis instead of the X axis. Once I had done this I was able to program some patches and make a short demo. I treated each patch as if it were a "note" in a sequence and used the SPG as a sequencer. This was interesting because it enabled me to have a quickly changing but repetitive sequence where I could play the controls of the synthesizer to change the timbre and tuning. Because the patch-pin matrix works parallel to the SPG, I was still able to patch the EMS joystick to control various aspects of the sound.

What was interesting to note in all of these examples was the effect that the non-buffered patching of the SPG had. As more connections are made from an input to various outputs, the signal spread over the outputs

is reduced in amplitude. When I was adding layers of feedback, this had a useful effect, slowly reducing the overall amplitude so that the volume was kept under control. However, when the Difas was used to route voltages to change pitches for instance, adding more outputs reduced the control voltages and changed the resulting pitches. I think that when using a SPG in a control-voltage based studio (rather than just patching signal paths) buffering to keep the output voltages identical to the inputs will be essential. The "clicks" created by patching signals at an arbitrary moment I exploited as musical material, but I can see that for some applications a very short cross-fade would be more suitable!"

Justin also shared some recordings made with his EMS VCS3 synth, which can be listened to following this link: https://soundcloud.com/user-520335425/synthi-spg-01-edit

He uses the SPG to rhythmically switch between the patches as a core of his composition. He also clearly notes that if more inputs are split to several outputs, the amplitude of the split signal is reduced and that the audible clicks in some applications would need a short cross-fade.

Christos Loupis (Sonology 1st year master)

Christos his master-research deals with 'coupled feedback systems as compositional tools' and he applied the SPG as a routing-tool for his feedback system. For the recordings he shared with me, no explicit description was added on how he uses the SPG in his setup. Since he is currently building one VC-SPG for his own setup, he is definitely interested in the new dimensions the SPG offers for his future compositions.

https://soundcloud.com/user-520335425/matrilex-cv-switch-christos-loupis

8.3 **Software results**

The data that has to be sent to the SPG in order to drive and control the device, is a properly formatted the right OSC-message, described in chapter "SPG communication" on page <$p>.

/pa [byte 0] [byte 1] … [byte 8] (activate the preset immediately)
/st [byte 0] [byte 1] … [byte 8] (store the preset in local memory)
/cf [byte0] (control the SPG with the configuration byte)

The Max/Msp patch I created and that is available for every user that wants to explore the SPG, is my implementation of how to interact with the SPG but there are many other ways to do so (figure 4.8 on page 32) because other software can be developed to send the OSC-message and control the SPG. During this master-research period multiple users developed new software tools and ideas for driving the SPG. In this section two examples.

Sohrab Motabar (Sonology 2nd yearMaster)

During the workshop Sohrab experimented with the original Max/Msp patch and added new interesting features like converting a drawing into a matrix preset and using noise to generate random patterns. See figure 8.4 to get an impression. Controlling the SPG using noise can generate interesting sound material when it is part of a synthesizer setup and 'drawing a new preset' introduces a new way of composing sounds - sound painting.

In any artistic project it is interesting to have access to random processes available - the matrix variations Sohrab introduced in the workshop can be of great help and inspiration for future SPG users.

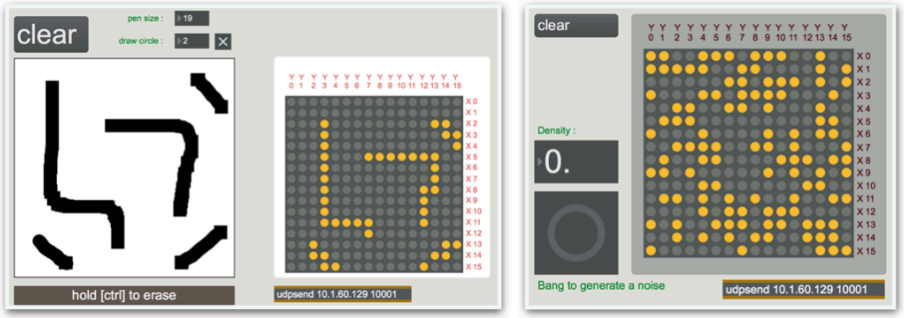


Figure 8.4 Left shows the graphical conversion to preset and on the right the noise to random preset.

Johan van Kreij (Sonology teaching staff)

One of the Max/Msp specialists, Johan van Kreij assisted me with the design of the Max/Msp patch for driving the SPG. Due to his interest in new technology he programmed a dedicated Max-object for the SPG. This is an object that generates the /pa OSC-message based upon Java script coding. This code is much more efficient in generating the OSC-messages in comparison with the conversions I made with the standard matrix-object available in Max/Msp. The standard matrix-object generates row- and column data with the corresponding value of the cell. If a cell is selected, the output generates [column-number], [row-number], [0 if it is off and 1 if it is on]. This series of three numbers has to be converted into the right syntax OSC-message for the SPG, as explained on page 43.

To achieve this conversion from the standard matrix object into the right OSC syntax with the standard Max/Msp objects is for many users a complex task. To simplify working with the SPG, Johan created a special object that generates the right OSC-message syntax for the SPG without any conversion needed. Catch a glimpse of the code on the right of figure 8.5 and on the left the basic Max/Msp patch linked to it.

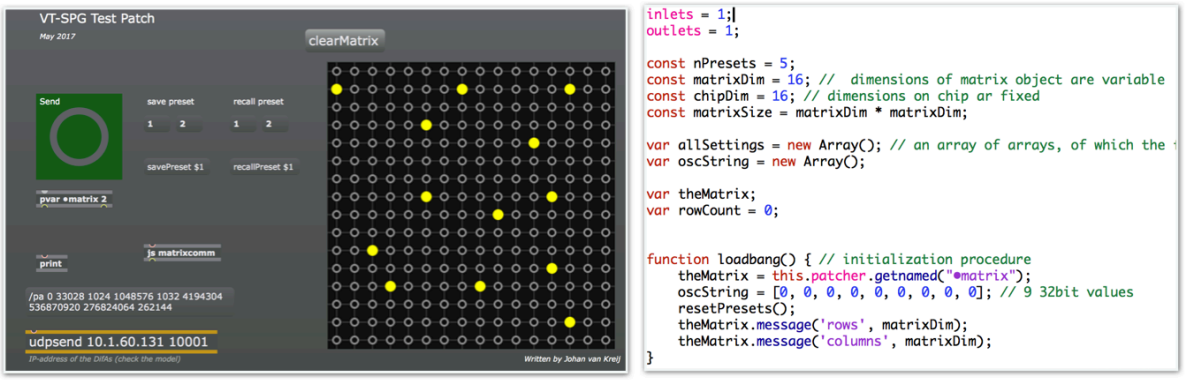


Figure 8.5. On the right a sample of the code and on the left the linked Max/Msp patch

9 **Conclusion**

What started two years ago as a project to design a tool for storing presets of patch cables in an analogue studio environment, turned out to be a more powerful instrument than I foresaw. To be able to drive 256 switches at high speeds, controlled by computer or controlled by external signals, is not only a signal routing system with memories, but it also introduces a new compositional tool or instrument for all kind of musical and artistic applications.

The VC-SPG introduces new possible approaches for sound generation and music composition when working in an analogue studio environment. Changing connections of patch cables, at high or low speeds, was actually never part of an analogue studio setup before and now can be applied even in sync with music, rhythmical patterns or triggered by external events.

The users of the VC-SPG clearly state that the interface introduces a new tool for their electronic music, workflow and composition. To be able to switch between presets during a live performance creates a more diverse range of sounds, or change the active preset of the studio setup by means of control-voltages, adds a new dimension to the artistic process.

The artistic results, shown in this thesis, are just the first results and I'm convinced the VC-SPG will be used more often in the creation of new electronic music and art-installations. I have identified different roles of the VC-SPG, but this does not mean that all possible roles have been identified so far.

This research-project enriched my own perception of electronic music by reading about electronic music, following lessons, discussing new options and attending the weekly sonology master-circle and listening to and participating in discussions. I am more involved with the musical and artistic application of the technology, than I was before. I explored the possibilities of the VC-SPG myself by making recordings using the interface embedded in the Doepfer A-100 modular synth. I enjoyed the exploration of sounds and the enrichment the VC-SPG introduced for experimenting with high-speed, low-speed, rhythmic, or chaotic preset changes and patterns.

One of the difficult challenges with the introduction of a new interface, like the VC-SPG, is to overcome its complexity (the multitude of possibilities). For a lot of users the SPG will be quite complex to use or apply because of dealing with network settings, dealing with a Max/Msp patch and making a choice of which signals to use for switching. It is therefore very important to keep the interface, as well as the software, as transparent and simple as possible. The strength and power of a new piece of technology lies in its simplicity — the easier the use, the better the interface.

I also think the introduction of the VC-SPG into the Sonology analogue studio Bea-5, in combination with the other OSC related equipment (OSC-CV and CV-OSC) could be

a start of a new series of workshops where Digital meets Analogue. By teaching and sharing my expertise it will eventually generate more artistic and musical results and stimulate the development of new technical designs in the future. I would therefore like to introduce more workshops about this subject and I am convinced that the younger and more digitally aligned students will eventually create completely new music with the aid of these OpenSoundControl interfaces.

9.1 **Future technical ideas and recommendation**

After this master research project, the development of the VC-SPG will continue, as it is an important part of my regular job within the Royal Conservatoire to develop new technology for Art and Education. There are still a lot of technical things to be improved and changed.

A list of ideas and suggestions:

- Make the sequence speed of the VC-SPG faster than the current limit of 1,7 kHz. It would be a great step forward to use a new 16-bit or 32-bit controller instead of the 8-bit microcontroller used in this design. The speed limit of the AD75019 audio-matrix itself is set to 20 kHz — that is fast enough.

- Solve the issue of the amplitude reduction when a signal is split. Splitting a signal to be routed to more outputs, should not have any influence on the amplitude. An additional electronic solution should be added to avoid this.

- Design a solution to get rid of the audible clicks when switching between presets. This might be done by working with two matrixes and using a fast fade in- and fade out circuit in between the outputs of the two matrices. This will be an interesting future challenge.

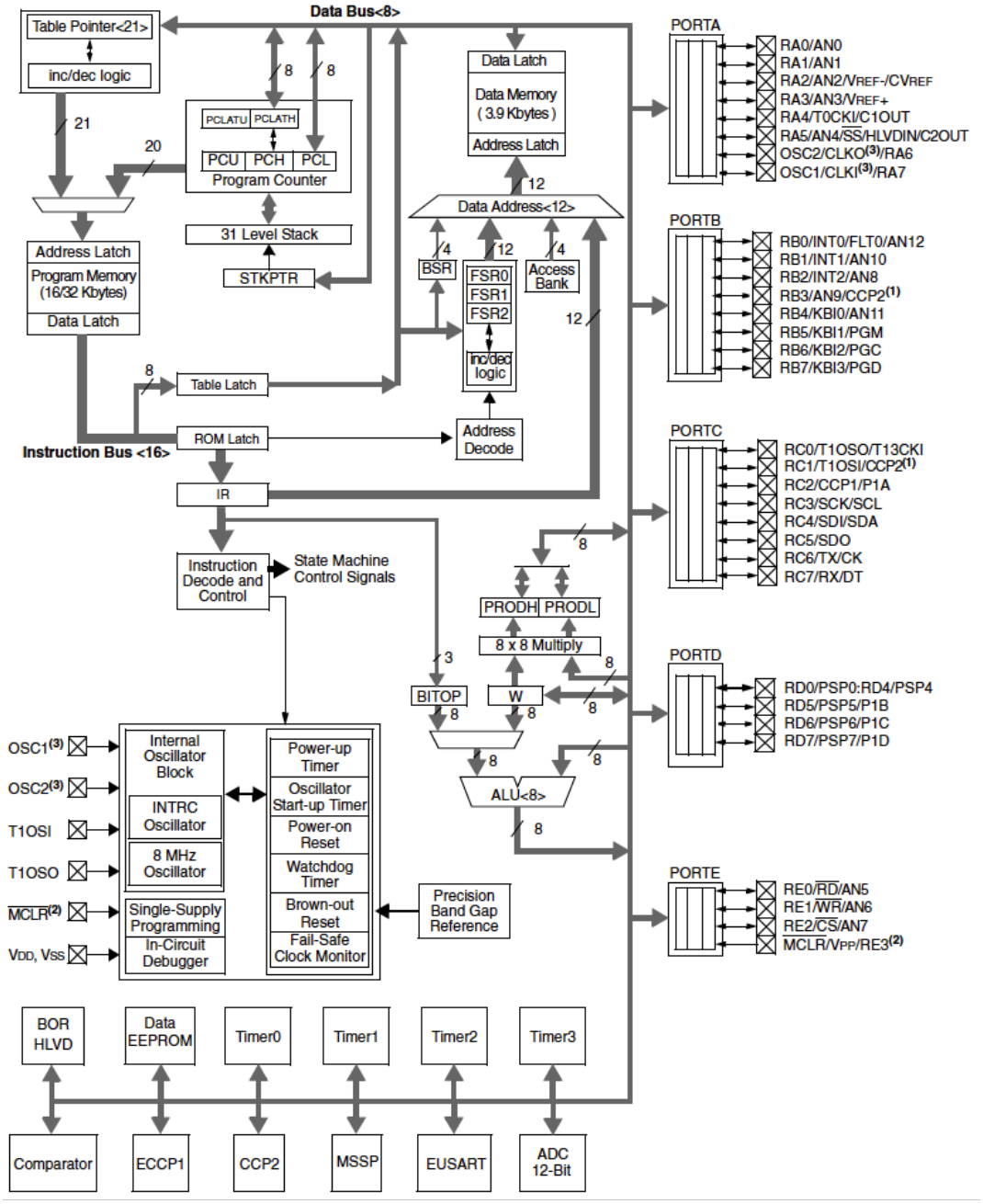
- A re-design of the printed circuit board is needed, so it will be less work to assemble a Eurocard format module.

- An update for the assembly-code. There are always better ways to optimize the code and to re-configure (simplify) the current setup. Furthermore additional code can be included that provides the possibility to store the presets, even when the SPG is powered down.

- An introduction of a so called 'hand-shake' option. When the user uploads the preset into the VC-SPG, a signal should be available that confirms the new presets are stored and ready to be used.

- Design a version to be installed into analogue studio Bea-5 in combination with clear instructions, so the usage of the VC-SPG will be less difficult and more accessible.

Microchip PIC18F2523 Block diagram



Appendix B

Reset switch	MCLR	1	PIC18F2325	28	B7	ShiftClck
Control Voltage In	A0, AN0	2		27	B6	StorageClk
Manual/On/Off	A1	3		26	B5	Data
Sequence On/Off	A2	4		25	AN11,B4	CV-preset In
OSC out On/Off	A3	5		24	B3	Next
Up/Down sequence	A4	6		23	B2	Previous
OSC/CV timing	A5	7		22	B1	Ext trigger IN
Power	GND	8		21	B0	Ext trigger IN
Xtal	OSC1	9		20	VCC	Power
Xtal	OSC2	10		19	GND	Power
Data	C0	11		18	C7	Rx Eusart
Shift clock	C1	12		17	C6	TX Eusart
Storage Clock	C2	13		16	C5	Both F/R
Int/Ext Sync	C3	14		15	C4	CV-Preset

Pin 1(MCLR) This is the master-clear pin. In normal function, this pin is connected to the +5V (or VCC). When this pin is connected to ground 0V (GND), the firmware running inside the controller will restart the program from the top. A hardware reset like this will take only a few micro-seconds.

Pin 2 (RA0) RA0 is defined to be a analogue input and the value connected to this pin, varying between 0V and 5V, is converted into a 12-bit number (0-4096). In the desgin of the SPG this pin converts the incomming CV to the speed of the sequence.

Pin 3 - Pin 7 (RA1 - RA5) are directly connected to switches changing the value of the pin between +5V and 0V.

Pin 8, Pin 19 and Pin 20 The power-supply connections. Vcc is +5V and GND, consisting of two connections, is 0V.

Pin 9, Pin 10 (OSC1, OSC2) The microcontrller need a clock signal ro 'run'. The quartz crystal oscillator connected to this pin provides the microcontroller with a very precize clock frequency of 7,342 MHz.

Pin 11 - Pin 13 (RC0 - RC2) The SPI communication protocol is used with these three pin's, to transfer data to the switching matrix. For details see the chapter communication.

Pin 14 - Pin 16 (RC3 and RC5) can be connected to switches if needed. Since version v85 of the firmware, the swithes can be overruled by the /cf osc-message and the hardware switches are optional.

Pin 16 and Pin 17 (RC6 and RC7) These pins communicate with the Xport with RS-232 (see communications)

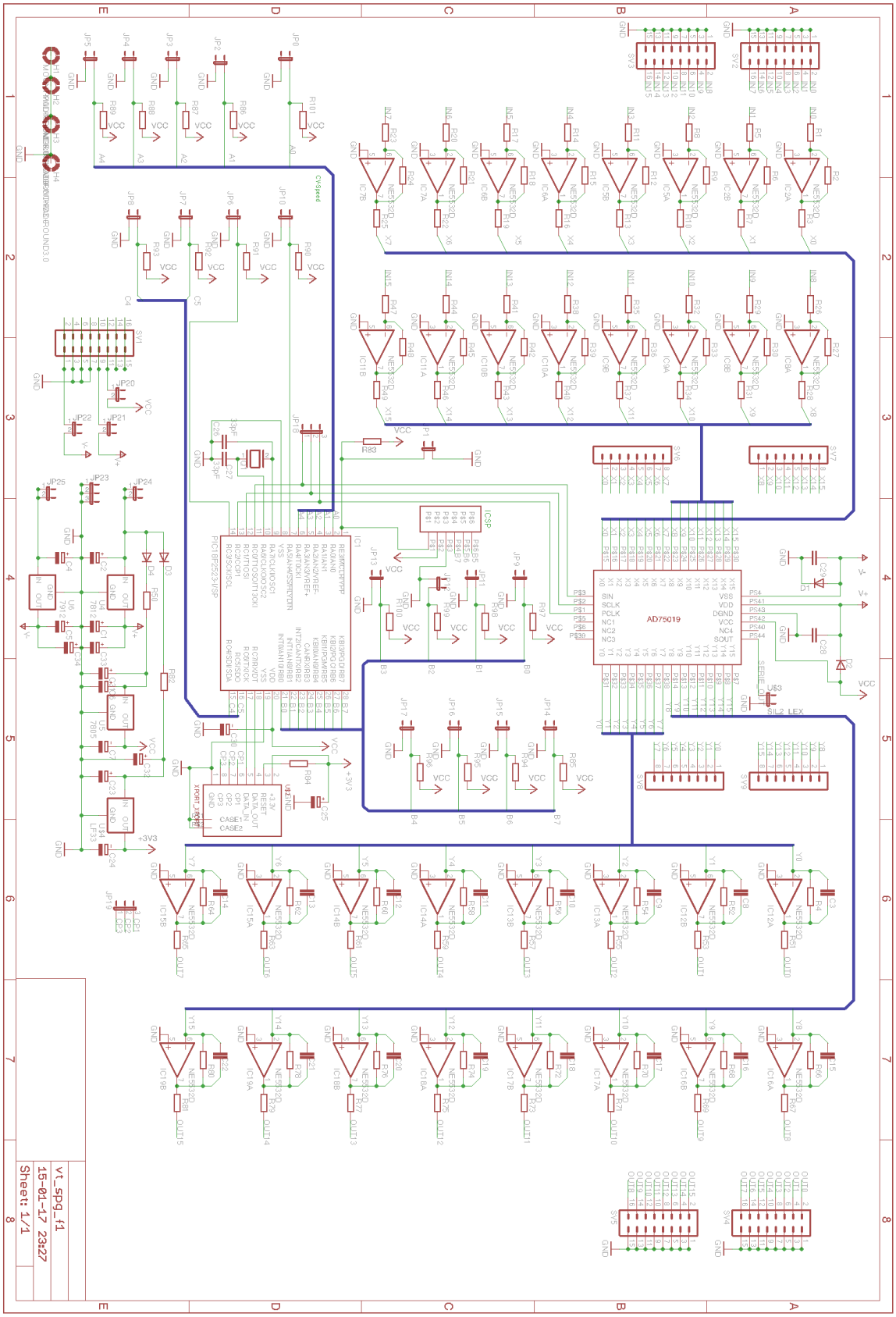
Pin 21 and Pin 22 (RB0 and RB1) The external trigger is connected to both of these pins and both are configured to generate an internal interrupt if the signals changes its values from 0V to 5V (rising edge) or from 5V to 0V (falling edge). An hardware interrup, or external interrupt, is an interruption of the firmware running and will execute a special interrupt routine. See the chapter Software for more detailed information.

Pin 23 and Pin 24 (RB3 and RB4) when the SPG is used in manual mode, these pins, connected to two push buttons, will step Up or Down through the presets stored in local memory and acitvate the preset.

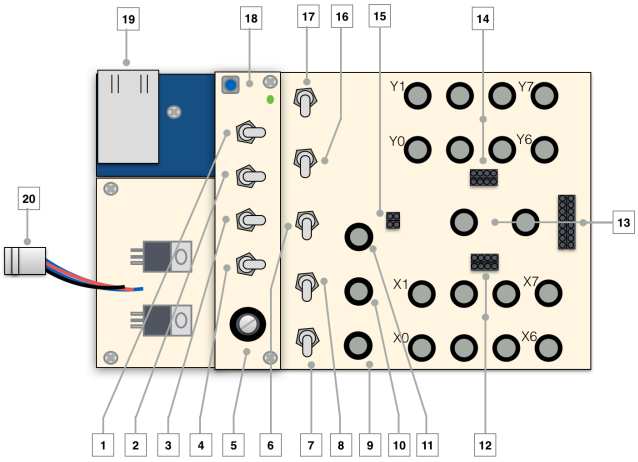
Pin 25 (RB4) Like pin 2, also this pin is internally connected to the ADC convertor. It will read an analogue voltage (ontrol-voltage) and convert this to a 12-bit value. The value is used to jump to a preset stored in local memory - so the value is converted to a preset-id.

Pin 26 - Pin 28 (RB5 - RB7) Like RC0-RC2, these pins use the SPI protocol to write the actual preset-id to the 5 led's, when the SPG is in manual mode.

VT-SPG full circuit (Eagle 7.4.0)



Reference Model 1
8 in, 8 out with banana, female-headers and physical switches

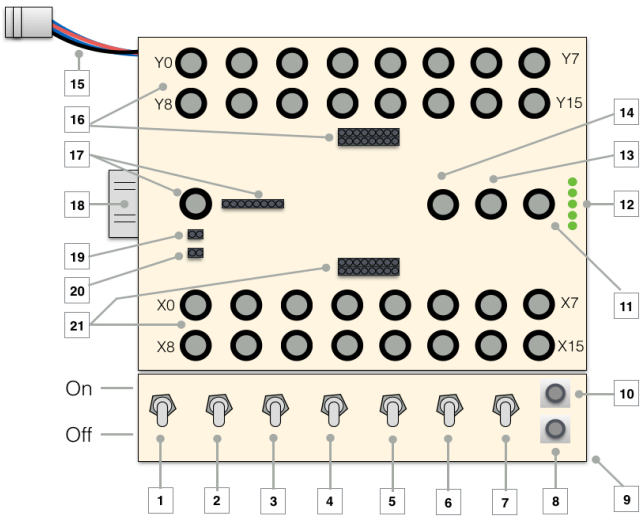


The first prototype of the SPG, model 1

1. **Sequence Up/Down**. This switch determines if the sequence is running up- or downwards.
2. OSC send messages On/Off. When you are sequencing through the local presets, OSC-messages will be send back to the computer when this switch is On. For debugging or visual feedback, this function is very useful. At high speeds this function will influence the performance.
3. Start/ Stop sequence. Start or Stop the sequence. When this button is switched On, the SPG will run through the presets stored in local memory.
4. Manual mode. If switched on, the SPG is in manual mode. In this (prototype model) the up/down push buttons are not included.
5. Control Voltage to speed. This potentiometer generates 0-5V for analog input A0 which converts the incoming voltage into sequence speed. Depending on the status of switch [8] this potentiometer or an external voltage can be the source.
6. Int./Ext. The SPG can sequence through presets based upon external or internal signals. When the switch is set to internal, the speed of the sequence is linked to the number (timer)received in the OSC-message. If the switch is set to external, the triggers of the sequence are determined by the edges of the external input [11].
7. CV/OSC. The speed of the sequence can be determined by a control voltage or by OSC (/cf message), which is stored in local memory.
8. CV speed Int./Ext. If this switch is set to external, the CV input of the SPG will be connected to an external control-voltage. When set to internal, the potentiometer described at [5] will be the voltage source.
9. CV-preset input. If you apply this control voltage and you select this option (v85), the value will determine the next preset if the SPG is in sequence mode. Switch [16] will activate or de-activate this mode.
10. CV-sequence speed input. This cv input can be used to change the speed go the sequence linked to the value of the control voltage. If the value of the cv is high, the speed is at its maximum (around 1,7kHz). Low values result in slower speeds.
11. **External Sync input**. This input can be used to connect an external (sync) signal. When switch [6] is set to external, the SPG will step to the next preset in sequence on the changing edge. See also switch [6].

12. X0-X7, Input. 8 x audio input on banana connector and small pin header. It is connected in parallel and connected in exact the same order as the banana connectors. The maximum amplitude the SPG can switch is between -12V and +12V.
13. GND. Both banana and pin-header can be used for the ground connection. When you are working with different devices, it's important to make good ground connections.
14. Y0-Y7, Output. These 8 banana-connectors are physical outputs of the SPG. Also the small pin-header connections can be used for output. It is connected in parallel and they are connected in exact the same order as the banana connectors.
15. +5V. To provide external electronics with +5V, this small pin connection can be used. Also connect the ground!
16. CV-preset On/Off. This switch determines if the cv-preset mode is active or not. See also description [9].
17. Positive/Negative. If the SPG is triggered by an external signal [11], this switch determines if only the positive-flank of the signal (OnSet) is used to step to the next sequence, or both the rising and falling edge.
18. Reset. This is the reset button. It will reset the onboard microcontroller.
19. Ethernet In/Out. Direct ethernet connection (cat5) to the computer or network. The communication is realized with OpenSoundControl.
20. Power connection. This is the power-connector. The SPG needs +15V, -15V and +5V. Please use het appropriate power supply.

Reference Model 2
16 in, 16 out, banana connectors and physical switches



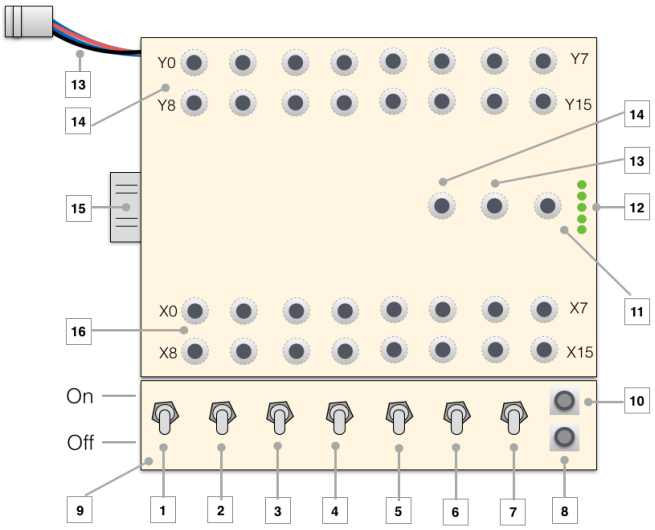
Model 2 with banana connectors and physical switches

1. Start/ Stop sequence.
   Start or Stop the sequence. When this button is switched On, the SPG will run through the presets stored in local memory.
2. OSC send messages On/Off. When you are sequencing through the local presets, OSC-messages will be send back to the computer when this switch is On. For debugging or visual feedback, this function is very useful. At high speeds this function will influence the performance.
3. Sequence Up/Down. This function determines if the sequence is running up- or downwards.
4. CV/OSC. The speed of the sequence can be determined by a control voltage or by OSC (/cf message), which is stored in local memory.
5. Int/Ext. The SPG can sequence through presets based upon external or internal signals. When the switch is set to internal, the speed of the sequence is linked to the number received in the OSC message. If the switch is set to external, the speed, or triggers of the sequence, is determined by the edges of the external input [13].
6. Positive/Negative. If the SPG is triggered by an external signal [11], this switch determines if only the rising edge of the signal (OnSet) is used to step to the next sequence, or both rising- and falling-edge.
7. Manual mode. If switched on, the user can step through the presets manually with switch [8, up] and switch [10, down].
8. Up. If manual mode is selected, pushing this button will step to the next preset.
9. Reset. Underneath the board (out of sight) a small push button is hidden. This is the reset button. It will reset the onboard microcontroller.
10. Down. If manual mode is selected, pushing this button will activate the previous preset.
11. External Sync. This input can be used to connect an external (sync) signal. When switch [5] is set to external, the SPG will step to the next preset in sequence on the changing edge. See also switch [6].
12. Led indication. These 5 leds indicate which preset is active when the SPG is in manual

mode [7]. The indication is binary, varying from 00000-11111 (=32 presets).

13. CV-preset input. If you apply this control voltage and you select this option (v85), the value will determine the next preset in line if the SPG is in sequence mode. Model 1 does not have a physical switch to activate this mode.
14. CV-sequence speed input. This cv input can be used to change the speed go the sequence linked to the value of the control voltage. If the value of the cv is high, the speed is at its maximum (around 1,7kHz). Low values result in slower speeds.
15. Power connection. This is the power-connector. The SPG needs +15V, -15V and +5V. Please use het appropriate power supply.
16. Y0-Y15, Output. These 16 banana-connectors are physical outputs of the SPG. Also the small pin-header connections can be used for output. It is connected in parallel and they are connected in exact the same order as the banana connectors. From left to right Y0-Y7 upper row and Y8-Y15 on the lower row.
17. GND. Both banana and pin-header can be used for the ground connection. When you are working with different devices, it's important to make good ground connections.
18. Ethernet In/Out. Direct ethernet connection (cat5) to the computer or network. The communication is realized with OpenSoundControl.
19. +5V. To provide external electronics with +5V, this small pin connection can be used.
20. +12V. To provide external electronics with +12V, this small pin connector can be used.
21. X0-X15, Input. 16 x audio input on banana connector and small pin header. It is connected in parallel and connected in exact the same order as the banana connectors. From left to right X0-X7 upper row and X8-X15 on the lower row. The maximum amplitude the SPG can switch is between -12V and +12V.

Reference Model 3
16 in, 16 out, minijack connectors and physical switches



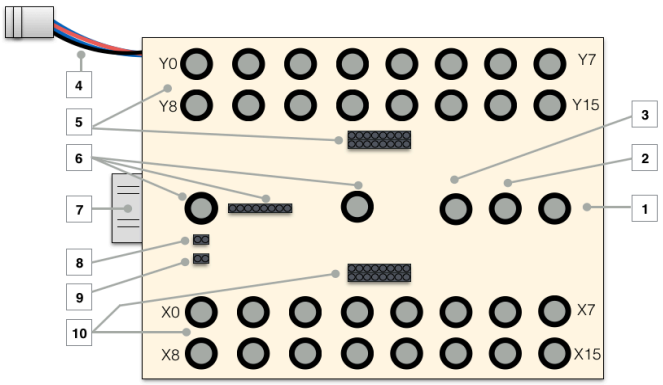
Model 3 with mono-minijack connectors for modular synths

1. **Start/ Stop sequence.** Start or Stop sequence through local memory. When this button is switched On, the SPG will run through the presets stored in local memory.
2. **OSC send messages On/Off**. When sequencing through the local presets OSC-messages will be send to the computer when this switch is On. At high speeds this function will influence the performance.
3. **Sequence Up/Down.** This function determines if the sequence is running up- or downwards.
4. **CV/OSC.** The speed of the sequence can be determined by a control voltage or by a OSC received number, which is stored in local memory.
5. **Int/Ext.** The SPG can sequence through presets based upon external or internal signals. When the switch is set to internal, the speed of the sequence is based upon the number received in the OSC message. If the switch is set to external, the speed of the sequence is determined by the edge of the external input (13).
6. **Positive/Negative.** If the SPG is triggered by an external signal (11), this switch determines if only the rising edge of the signal (OnSet) is used to step to the next sequence, or both the falling and rising edge.
7. **Manual mode.** If switched on, the user can step through the presets manually with switch (8, up) and switch (10, down).
8. **Up.** If manual mode is selected, pushing this button will step to the next preset.
9. **Reset.** Underneath the board (out of sight) a small push button is hidden. This is the reset button. It will reset the onboard microcontroller.
10. **Down.** If manual mode is selected, pushing this button will step to the previous preset.
11. **External Sync.** This input can be used to connect an external (sync) signal. When switch (5) is set to external, the SPG will step to the next preset in sequence on the changing edge. See also switch (6).
12. **Led indication.** These 5 leds indicate which preset is active when the SPG is in manual mode (7). The indication is binary, varying from 00000-11111 (32 presets).

13. **Power connection.** This is the power-connector. The SPG needs +15V, -15V and +5V. Please use het appropriate power supply.
14. **Y0-Y15, Output.** These 16 banana-connectors are physical outputs of the SPG. Also the small pin header can be used for output. It is connected in parallel and the connected in exact the same order as the banana connectors. From left to right Y0-Y7 upper row and Y8-Y15 on the lower row.
15. **Ethernet In/Out.** Direct ethernet connection (cat5) to the computer or network. Communication is realized with OpenSoundControl.
16. **X0-X15, Input.** 16 x audio input on banana connector and small pin header. It is connected in parallel and connected in exact the same order as the banana connectors. From left to right X0-X7 upper row and X8-X15 on the lower row. The maximum amplitude for the SPG to switch is between -12V and +12V.

Reference Model 4

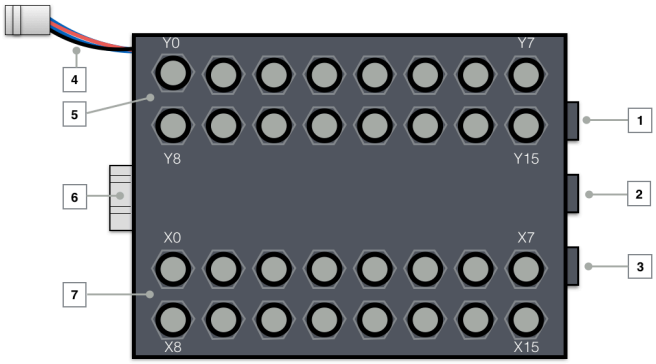
16 in, 16 out, banana-connectors + pin headers, no switches



Model 4, with banana connections and pin-headers. No switches.

1. **Ext. Trigger input.** Sending trigger signals with falling or rising edge, will activate the step to the next preset, when the SPG is in sequence mode.
2. **CV-preset input.** If you apply this control voltage and you select this option (v85), the value will determine the next preset in line if the SPG is in sequence mode. Model 1 does not have a physical switch to activate this mode.
3. **CV-sequence speed input.** This cv input can be used to change the speed go the sequence linked to the value of the control voltage. If the value of the cv is high, the speed is at its maximum (around 1,7kHz). Low values result in slower speeds.
4. **Power connection.** This is the power-connector. The SPG needs +15V, -15V and +5V. Please use het appropriate power supply.
5. **Y0-Y15, Output.** These 16 banana-connectors are physical outputs of the SPG. Also the small pin-header connections can be used for output. It is connected in parallel and they are connected in exact the same order as the banana connectors. From left to right Y0-Y7 upper row and Y8-Y15 on the lower row.
6. **GND.** Both banana and pin-header can be used for the ground connection. When you are working with different devices, it's important to make a good ground connection.
7. **Ethernet In/Out.** Direct ethernet connection (cat5) to the computer or network. The communication is realized with OpenSoundControl.
8. **+5V.** To provide external electronics with +5V, this small pin connection can be used.
9. **+12V.** To provide external electronics with +12V, this small pin connector can be used.
10. **X0-X15, Input.** 16 x audio input on banana connector and small pin header. It is connected in parallel and connected in exact the same order as the banana connectors. From left to right X0-X7 upper row and X8-X15 on the lower row. The maximum amplitude the SPG can switch is between -12V and +12V.

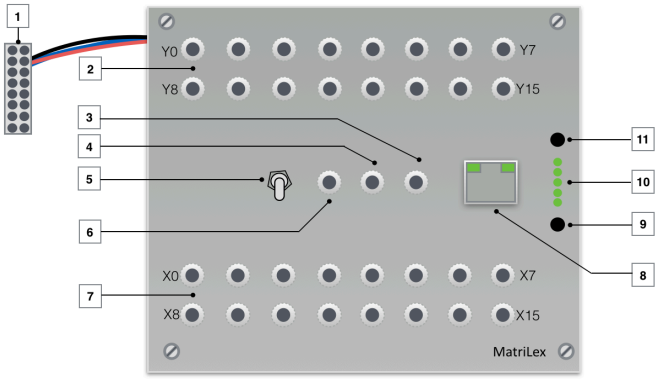
Reference model 5
16 in, 16 out, 1/4" jack connectors and no physical switches



Model 5 made with 1/4" inch jack (PL)

1. **Ext. Trigger input.** Sending trigger signals with falling or rising edges, will activate the step to the next preset, when the SPG is in sequence mode.
2. **CV-preset input.** If you apply this control voltage and you select this option (v85), the value will determine the next preset in line if the SPG is in sequence mode. Model 1 does not have a physical switch to activate this mode.
3. **CV-sequence speed input.** This cv input can be used to change the speed go the sequence linked to the value of the control voltage. If the value of the cv is high, the speed is at its maximum (around 1,7kHz). Low values result in slower speeds.
4. **Power connection.** This is the power-connector. The SPG needs +15V, -15V and +5V. Please use het appropriate power supply.
5. **Y0-Y15, Output.** These 16 banana-connectors are physical outputs of the SPG. Also the small pin-header connections can be used for output. It is connected in parallel and they are connected in exact the same order as the banana connectors. From left to right Y0-Y7 upper row and Y8-Y15 on the lower row.
6. **Ethernet In/Out.** Direct ethernet connection (cat5) to the computer or network. The communication is realized with OpenSoundControl.
7. **X0-X15, Input.** 16 x audio input on banana connector and small pin header. It is connected in parallel and connected in exact the same order as the banana connectors. From left to right X0-X7 upper row and X8-X15 on the lower row. The maximum amplitude the SPG can switch is between -12V and +12V.

Reference model 6
16 in, 16 out, minijack connectors and one manual-mode switch

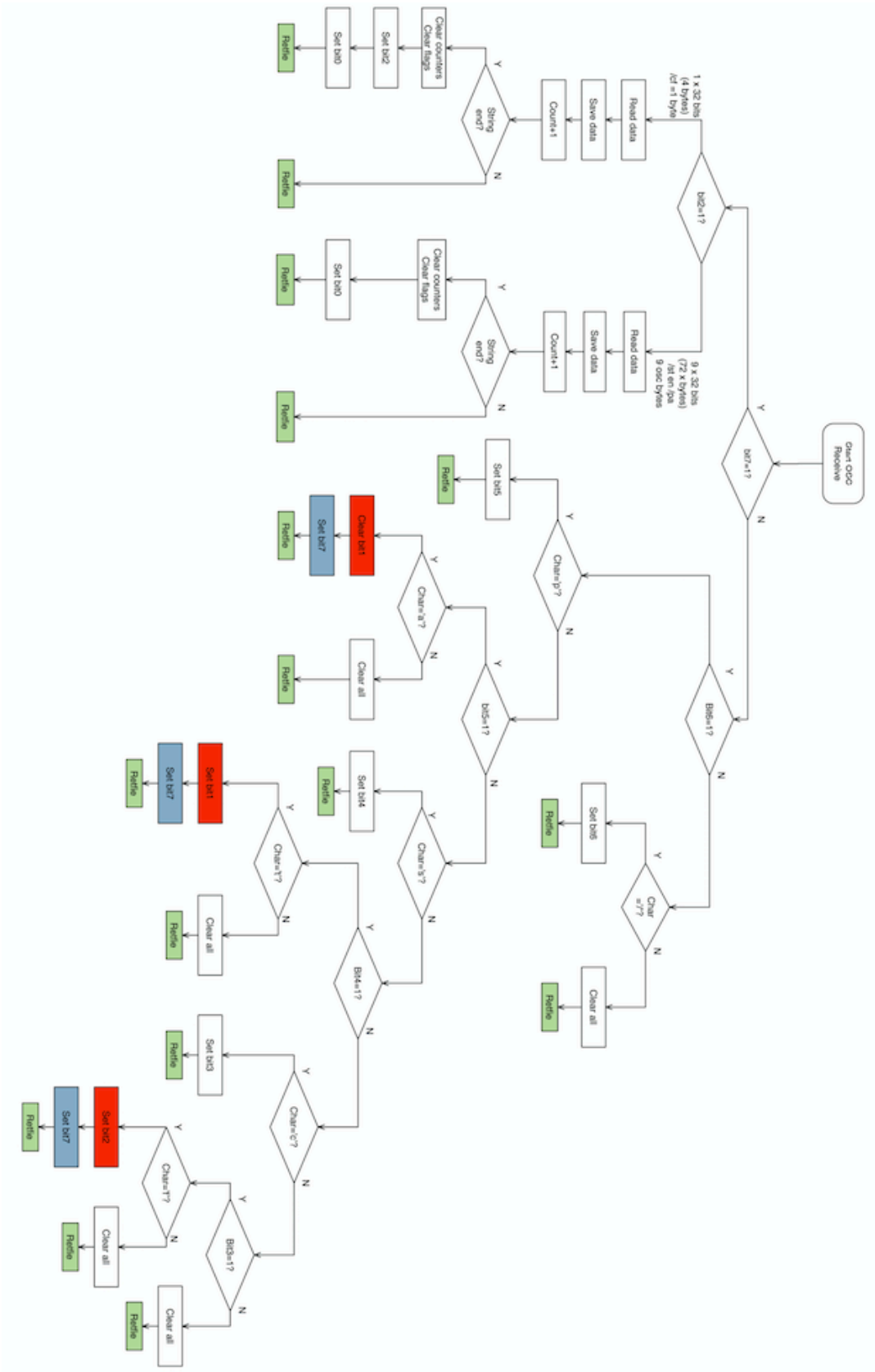


Model 6, eurorack compatible

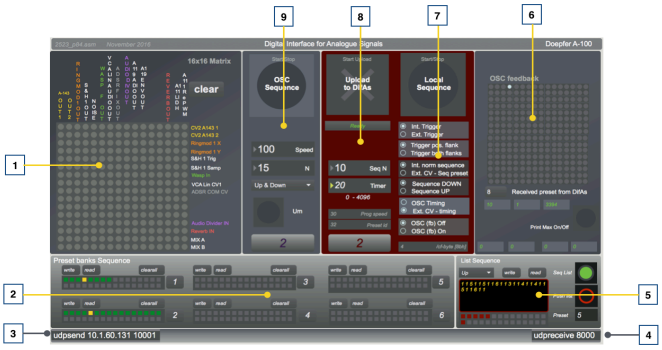
1. **Euro-rack power supply.** This is the standard power-supply connector for Doepfer related Euro-rack models. It consists of +12, -12, Gnd, +5 and two bus connections. The SPG does not support the bus-connections (yet). The exact pin-out can be found on the Doepfer website.
2. **Y0-Y15, Output.** These 16 banana-connectors are physical outputs of the SPG. Also the small pin-header connections can be used for output. They are connected in parallel and connected in exact the same order as the banana connectors. From left to right Y0-Y7 upper row and Y8-Y15 on the lower row.
3. **CV-preset input.** If you apply this control voltage and you select this option (v85), the value will determine the next preset in line if the SPG is in sequence mode. Model 1 does not have a physical switch to activate this mode and switch between internal potentiometer and external cv.
4. **CV-sequence speed input.** This cv input can be used to change the speed go the sequence linked to the value of the control voltage. If the value of the cv is high, the speed is at its maximum (around 1,7kHz). Low values result in slower speeds.
5. **Manual On/Off.** You can switch to manual-mode. The leds [10] will indicate which preset is active and with the switches [9] and [11] you can step Up or Down.
6. **External trigger input.** Sending trigger signals with falling and/or rising edges, will activate the step to the next preset, when the SPG is in sequence mode.
7. **X0-X15, Input.** 16 x audio input on banana connector and small pin header. It is connected in parallel and connected in exact the same order as the banana connectors. From left to right X0-X7 upper row and X8-X15 on the lower row. The maximum amplitude the SPG can switch is between -12V and +12V.
8. **Ethernet In/Out.** Direct ethernet connection (cat5) to the computer or network. The communication is realized with OpenSoundControl.
9. **Up.** If manual mode is selected, pushing this button will step to the next preset.
10. **Led indication.** These 5 leds indicate which preset is active when the SPG is in manual mode [7]. The indication is binary, varying from 00000-11111 (=32 presets).
11. **Down.** If manual mode is selected, pushing this button will activate the previous preset.

Appendix E

Interrupt routine OSC-message receive



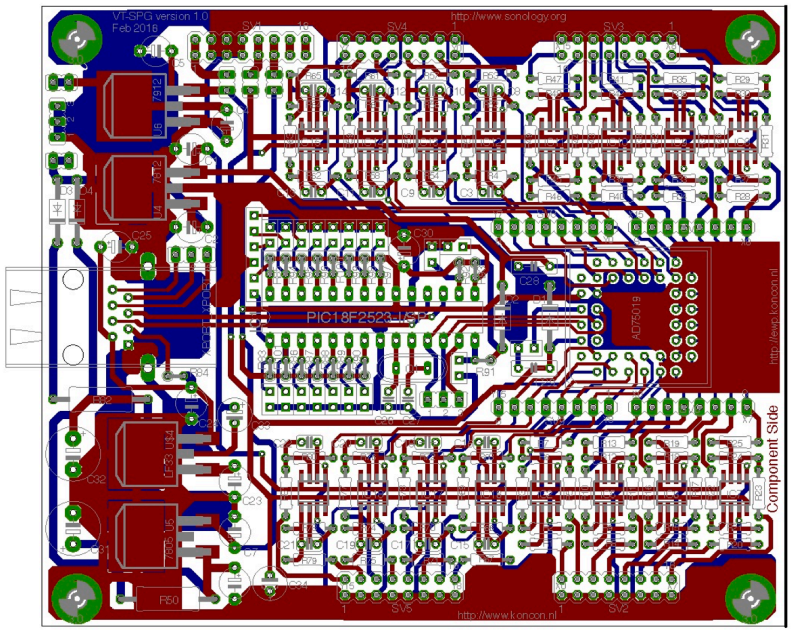
Max/Msp patch (v85)



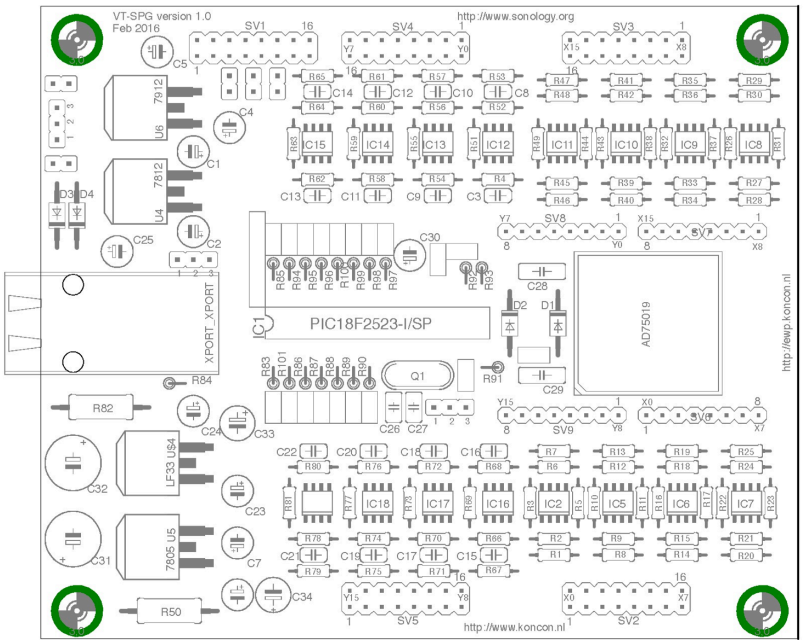
This is maxpatch version v85 and can be downloaded at: http://www.ipson.nl/osc-downloads

1. **The matrix.** Clicking on the gray dots wil instantly send a /pa string to the SPG, activating this connection. If a preset is created, this preset can be stored by shift-clicking on one of the locations in [2].
2. **Preset banks.** In this box the presets created in the matrix [1] can be stored. The preset can also be written to file or read from file. To create a physical connection, click on the right junctions in the matrix[1]. If the dots are lighting up in yellow, the connections are active and the preset can be stored. Storing a preset can be done by clicking on the right number preset box in [2] and hold down the shift key at the same time. When all presets are made and it is time to sequence through the different presets, select the right number (1-6) in box [2] and start the sequence with the start button in box [8].
3. **Udpsend.** The computer has to send the OSC-messages to the right ip-address and port. On the left side in box[3] the ip-address and port of the SPG has to be filled in. The port number should be 10001.
4. **Udpreceive.** On the right side the receiving port is defined. **Port 8000** will be set to receive OSC-messages from the SPG, unless it is configured differently in the embedded webserver (see getting started).
5. **Sequence from lists.** When you created presets, but you want to sequence through the presets in a specific order other than up/down, you should create and use lists. If the button 'Seq List' is active (green, or blue) the OSC sequence start/stop [9] will sequence from the list [5]. In this list presets-id's are written from the selected bank at [2]. Before the list can be sequenced, the list should be 'pushed' or activiated (red/black button).
6. **OSC visual feedback.** This part of the patch shows the incoming OSC-messages from the SPG in order to have (visual) feedback. Whether the SPG will send OSC messages back to the computer depends on the settings of the switches or the settings of the controls explained at [6]. When the 'print Max' switch is turned on, the patch will print the vales that are send to the SPG in the max-window. This option is for controlling or testing purposes only - it will slow down the total performance of the Max patch considerably.
7. **Configure SPG.** This part of the patch controls the configuration. The switches in this box replacing the physical switches on the different models. The biggest button on top start/stops the local sequence. The following options can be chosen going down in the patch: Internal or external trigger; Only positive or falling and rising edge reaction for external trigger; Internal normal sequence (up/down) or ext. cv-sequence; At normal sequence up or down; The source for the speed (timing) from OSC (the patch) or from ext. Cv-speed; The last one is switching On/Off the OSC-feedback.
8. **Uploading presets to the SPG.** This box sends the /st OSC-message in order to store the presets in local memory. When the big cross (upload to SPG) is clicked, the active preset bank, selected at [2], will be uploaded. The rest of the variables shown in this box [7] generate a new /cf OSC-message changing the timer and the amount of presets (N) of the sequence.
9. **OSC-sequence.** This part starts or stops the OSC-sequence. If it is active the patch sends /pa OSC-messages at variable speeds and it sequences through the selected preset-bank [2]. The speed and the amount of presets of the sequence (N) can be set as well. To offer more variation a choice of Up/Down, Pendulum (up and down) or 'Urn' are offered. This last option 'Urn' sequences through a set of presets (bank) in random order, but will use all the presets at least one time before starting with the next round.

VC-SPG Printed Circuit Board layout (PCB) (Eagle 7.4.0)



Printed circuit board with all layers visible. Red is the top-layer. Blue is the bottom-layer.



Printed circuit board with only the components (Silk screen tekst layer)

```
;_____________________________________________________________
; processor and configuration
;_____________________________________________________________

;kristal 7,3728 Mhz / baudrate van 921600 K and a clock of 29.491 MHz

        INCLUDE "p18f2523.inc"      ;specificatie processor
        radix dec                   ;default is decimal
        config  OSC = HSPLL         ;HS oscillator, PLL enabled (Clock Frequency = 4 x FOSC1)
        config  WDT = OFF           ;WDT watch dog timer
        config  DEBUG = OFF         ;DEBUG background debugger (RB6 en RB7 in circuit debug)
        config  LVP = OFF           ;LVP Single-Supply ICSP in circuit programming (pin RB5)
        config  XINST = OFF         ;XINST extended instruction set (moet uit)
        config  BOREN = ON          ;BOREN brown-out-timer reset CPU als de spanning te laag wordt
        config  PWRT = ON           ;PWRT power up timer
        config  PBADEN = ON         ;PBADEN portb<4:0> digitaal i/o (OFF) analoog input (ON)
        config  CP0 = OFF           ;Code Protection Block 0 --> Disabled
        config  CP1 = OFF           ;Code Protection Block 1 --> Disabled
        config  CPB = OFF           ;Boot Block Code Protection --> Disabled
        config  WRT0 = OFF          ;Write Protection Block 0 --> Disabled
        config  WRT1 = OFF          ;Write Protection Block 1 --> Disabled
        config  WRTB = OFF          ;Boot Block Write Protection --> Disabled
;-------------------------------------------------------------------------------------------------------------
; def. memory and interrupts vector
;-------------------------------------------------------------------------------------------------------------
        ORG     0x0000              ;start free memory / resort vector
        goto    init
        ORG     0x0008              ;interrupt high priority
        goto    hp_int
        ORG     0x0018              ;interrupt low priority
        goto    lp_int
;-------------------------------------------------------------------------------------------------------------
; definition variables
;-------------------------------------------------------------------------------------------------------------
time            equ 0x0020
recflag         equ 0x0021
teller          equ 0x0022
nummer          equ 0x0023
datacnt         equ 0x0024
tmprec          equ 0x0025
waarde          equ 0x0026
bitcnt          equ 0x0027
preset_id       equ 0x0028
temp            equ 0x0029
int_stat        equ 0x002a
t0telh          equ 0x002b
t0tell          equ0x002c
presteld        equ 0x002d
prestelu        equ 0x002e
a_preset        equ 0x002f
adh             equ 0x0030
adl             equ 0x0031
downteller      equ 0x0032
oscconfig       equ0x0033
telstat         equ 0x0034
var1            equ 0x0035
var2            equ 0x0036
var3            equ 0x0037
tmpdisp         equ 0x0038
ledcnt          equ 0x0039
tmppresh        equ 0x003a
tmppresl        equ 0x003b
prescvtel       equ 0x003c
cf_byte         equ 0x003d
SPG_stat        equ 0x003e
;-----------------------------------------------------------------------
;Initialization and definition
;-----------------------------------------------------------------------
init
        clrf    PORTA
```

```
        clrf    PORTB
        clrf    PORTC
        clrf    LATA
        clrf    LATB
        clrf    LATC
;Configure in- and output
        movlw   B'00111111'         ; a0 analog in, a1-a5 switch input,
        movwf   TRISA,              ;
        movlw   B'00011111'         ; portb,0 INT0; portb,1 INT1, rising/falling edge, cv-preset on B4
        movwf   TRISB               ;
        movlw   B'10111000'         ; portc7 Rx in; portc3 input
        movwf   TRISC
;-----------------------------
;Configure AD convertor
        movlw   B'00000000'         ; channel 0 An0 (porta,0) als input analog;
        movwf   ADCON0              ; Ad off
        movlw   B'00001110'         ; An0 and An11 input for CV
        movwf   ADCON1              ; b3-b0 (pcfg3-pcfg0) = 1110
        movlw   B'10110010'         ; format (right aligned)
        movwf   ADCON2              ; Acquisition time (b5-b3) 16Tad
                                    ; Conversion clock select bits (b2-b0) Fosc/32
; Interrupt for AD conversion
        bcf             pie1,6      ; interrupt enable (disable)
;-----------------------------
;instellen seriele interface (tbv xport)

        bsf     TXSTA,2             ;high speed (BRGH=1)
        bcf     TXSTA,3             ;sync break transmission completed
        bcf     TXSTA,4             ;asynchroon (SYNC=0)
        bsf     TXSTA,5             ;transmit enabled (TXEN=1)
        bcf     TXSTA,6             ;8 bits transmissie (TX9=0)
        bsf     RCSTA,4             ;enable continuous receiver (CREN=1)
        bcf     RCSTA,6             ;8 bits (RX9=0)
        bsf     RCSTA,7             ;serial port enable (SPEN=1)
        bcf     baudcon,7           ;ABDOVF no auto-baud acquisition rollover
        bsf     baudcon,3           ;8 bit baud rate (BRG16=0)
        bcf     baudcon,1           ;WUE wake-up enable bit
        bcf     baudcon,0           ;baud rate measurement disabled (ABDEN=0)
        movlw   D'7'                ;decimale waarde voor de baudrate bij kristal 7,3728 Mhz
                                    ;7 bij 921600K, 15 bij 460800K, 31 bij 230400K, 63 bij 115200K
        movwf   SPBRG
;-----------------------------
;instellen tmr0
        movlw   b'01001000'         ; tmr0 tmp off (b7), 8 bit timer, psa not assigned; pres scale: 1:1
        movwf   t0con
        bcf     t0con,7             ; timer0 on/off   (0=off, 1=on)
; interrupt van trm0
        bsf     intcon,5            ; interrupt enable tmr0
;-------------------------------------------------------------------------
; Config interrupt on recflag portb,0 (falling, rising edge)
        bcf     intcon,4            ; INT0IE (interrupt enable)
        bsf     intcon2,6           ; INTEDG0 bit. Als 1, rising edge. Als 0, falling edge
        bcf     intcon,1            ; INT0IF (interrupt flag)
;-------------------------------------------------------------------------
; Config interrupt on recflag portb,1 (falling, rising edge)
        bsf     intcon3,3           ; INT1IE Interrupt enable bit
        bcf     intcon2,5           ; INTEDG1 bit. Als 1, rising edge. Als 0, falling edge
        bcf     intcon3,0           ; INT1IF Interrupt flag

        bsf     intcon3,6           ; INT1IP Interrupt priority (set on high)
;Config interrupt

        bsf     pie1,5              ; enables/disables receive int.
        bcf     pie1, txie          ; disable tx int.
        bsf     Rcon,7              ; IPEN enable priority levels on interrupts (p.102)
        bsf     Rcon,7              ; power on reset (por) Zie p42.
        bsf     ipr1,5              ; RCIP EUSART receive interrupt priority (1=high, 0=low) (p.96)
        bsf     intcon2,2           ; Tmr0 interrupt high prio
        bsf     intcon2,7           ; portb internal pullup active
        bsf     INTCON,GIEH         ; GIE/GIEH global high priority interrupt enable
        bsf     INTCON,GIEL         ; PEIE/GIEL enables all low priority interrupts

;------------------
; Start values
```

```
        clrf    recflag
        clrf    oscconfig
        clrf    waarde
        clrf    int_stat
        clrf    t0tell
        clrf    t0telh
        lfsr    fsr0, 40h               ; init waarde receive routine
        movlw   d'8'
        movwf   bitcnt
        clrf    porta
        bsf     portc,2                 ; load moet hoog zijn

;-----------------------------------------------------
; Make AD75019 clean at startup
        movlw   d'32'                   ; 256 x 0 naar AD75019
        movwf   teller
init_empty                              ; AD75019 op nul zetten
        clrf    waarde                  ; 32 x 8 bits sturen
        call    writedata
        decfsz  teller
        goto    init_empty
        call    trigger                 ; maak actief

; Led-display off:
        movlw   0x00
        movwf   a_preset
        call    writeled
        goto    main

;++++++++++++++++++++
; Interrupts handling routines
;++++++++++++++++++++
hp_int
; ------------------------------------------------
; High Prior Interrupt. Afhandelen van de timer
; interrupt handling tmr0

        btfsc   intcon,2                ; timer0 interrupt?
        goto    metro                   ; Yes, go to metro
        btfsc   intcon,1                ; interrupt portb,0 (INT0)
        goto    sync_time_0             ; yes
        btfsc   intcon3,0               ; interrupt portb,1 (INT1)
        goto    sync_time_1             ; yes
;-----------------------------------------------------
osc
        btfss   pir1,5                  ; Received data eusart?
        retfie  fast                    ; No, return
osclezen
        btfss   recflag,7               ; ja, alle header info correct?
        goto    header_check            nee, dus eerst checken
        btfsc   recflag,2               ; is het /cf (bit2=1) of /st /pa (bit2=0)
        goto    cf_byte_rec             ; bit2=1
pa_st_byte
        movff   rcreg, postinc0         ; ja, inlezen naar indf0 en fsr0l verhogen
        movlw   0x73
        cpfsgt  fsr0l                   ; alles binnen?
        retfie  fast                    ; nee, wachten op volgende interrupt
        movlw   b'00000010'             ; /st:bit1=1, /pa:bit1=0, rest bits op nul
        andwf   recflag
        bsf     recflag,0               ; set flag voor main routine
        bcf     oscconfig,7             ; reset flag voor main routine
        retfie  fast
cf_byte_rec
        movff   rcreg, postinc0         ; Yes, write to indf0 and fsr0l +1
        movlw   0x8b
        cpfsgt  fsr0l                   ; All in?
        retfie  fast                    ; No, wait for next interrupt
        clrf    recflag
        bsf     recflag,0               ; set flag for main routine
        bsf     oscconfig,7             ; set flag for main routine
        retfie  fast
```

```
;------------------------------
header_check
movff   rcreg, tmprec           ; Read char
        btfsc   recflag,6       ; Busy receiving header?
        goto    b5check         ; Yes, check next
        movlw   a'/'            ; No, test char '/'
        cpfseq  tmprec
        retfie  fast
        bsf     recflag,6
        retfie  fast
b5check
        btfss   recflag,5       ; Char 'p' received?
        goto    b4check
        movlw   a'a'
        cpfseq  tmprec
        goto    clearall
        bsf     recflag,7
        bcf     recflag,1
        bcf     recflag,2
        lfsr    fsr0, 43h
        retfie  fast
b4check
        btfss   recflag,4       ; Char 's' received?
        goto    b3check
        movlw   a't'
        cpfseq  tmprec
        goto    clearall
        bsf     recflag,7
        bsf     recflag,1
        bcf     recflag,2
        lfsr    fsr0, 43h
        retfie  fast
b3check
        btfss   recflag,3       ; Char 'c' received?
        goto    pcheck
        movlw   a'f'
        cpfseq  tmprec
        goto    clearall
        bsf     recflag,7
        bcf     recflag,1
        bsf     recflag,2
        lfsr    fsr0, 83h
        retfie  fast
pcheck
        movlw   a'p'
        cpfseq  tmprec
        goto    scheck
        bsf     recflag,5
        retfie  fast
scheck
        movlw   a's'
        cpfseq  tmprec
        goto    ccheck
        bsf     recflag,4
        retfie  fast
ccheck
        movlw   a'c'
        cpfseq  tmprec
        goto    clearall
        bsf     recflag,3
        retfie  fast
clearall
        clrf    recflag
        clrf    oscconfig
        lfsr    fsr0, 40h
        retfie  fast
```

```
;-------------------------------------
metro
        btfsc   int_stat,7              ; busy?
        goto    bezig                   ; Yes
        movff   adh, t0telh             ; No, new values
        movff   adl, t0tell
        bsf     int_stat,7
        goto    intend
bezig
        tstfsz  t0telh
        goto    _tel
last_tel
        tstfsz  t0tell
        goto    lownotzero
        goto    tel_ready
_tel
        tstfsz  t0tell
        goto    lownotzero
        decf    t0telh
        movlw   0xff
        movwf   t0tell
        goto    intend
lownotzero
        decf    t0tell
        goto    intend
tel_ready
        bsf     int_stat,0              ; ready, set flag
        bcf     int_stat,7
intend
        bcf     intcon,2                ; clear int flag
        retfie  fast
;-----------------------------------------
sync_time_0                             ; interrupt for rising edge
        bsf     int_stat,0              ; set flag for main
        bcf     intcon,1                ; clear flag
        retfie  fast
sync_time_1                             ; interrupt voor falling edge
        bsf     int_stat,0              ; set flag voor mainroutine
        bcf     intcon3,0               ; clear flag
        retfie  fast
;---------------------------------------
; LOW prior interrupt routine
lp_int
        retfie
; _____________________________________________________________
; Main loop
;_____________________________________________________________
main
        btfss   oscconfig,7             ; flag voor nieuwe /cf ?
        goto    newdatacheck            ; no, check other data
        lfsr    fsr1,8bh                ; yes, read configdata
        movff   indf1, cf_byte          ; and store in cf_byte
        btfsc   SPG_stat,0
        bcf     oscconfig,7             ; clear flag
newdatacheck
        btfss   recflag,0               ; string ontvangen?
        goto    nonewdata               ; no, check manual and sequence
store_or_write
        btfsc   recflag,1               ; yes, is it a /st?
        goto    prog                    ; yes, transfer ontvangen byte
        call    wp0                     ; no, it's a patch /pa
        call    trigger                 ; activate
        clrf    recflag,0               ; clearflag
        goto    main                    ; back to main
nonewdata
        btfsc   porta,1                 ; manual knop actief?
        goto    mlstart                 ; ja, start manual mode
sw_check
        btfss   porta,2                 ; sequence knop actief?
        goto    cf_check
        bcf     oscconfig,0              ; main switch for config source choice
        goto    MasterMainSequence
cf_check
        btfss   cf_byte,6               ; check if source is cf_byte
```

```
        goto    main                    ; no, goto main
        bsf     oscconfig,0             ; yes, set main bit
;-----------------------------------------------
; Main Sequence Initialization
;-----------------------------------------------
MasterMainSequence
        btfss   oscconfig,0             ; check mode (sw or /cf)
        goto    sw_byte_config
cf_byte_config
        lfsr    fsr1,8bh                ; read configdata
        movff   indf1, cf_byte          ; and store in cf_byte

        movff   cf_byte, SPG_stat
        goto    sequence_init
sw_byte_config
        btfsc   porta,3                 ; porta,3 = OSC return On/Off
        goto    setb0
        bcf     SPG_stat,0
        goto    b1
setb0
        bsf     SPG_stat,0
b1      btfsc   porta,4                 ; porta,4 = Sequence Up/Down
        goto    setb1
        bcf     SPG_stat,1
        goto    b2
setb1
        bsf     SPG_stat,1
b2      btfsc   porta,5                 ; porta,5 = Timing source OSC/Cv
        goto    setb2
        bcf     SPG_stat,2
        goto    b3
setb2
        bsf     SPG_stat,2
b3      btfsc   portc,3                 ; portc,3 = Int / Ext trigger
        goto    setb3
        bcf     SPG_stat,3
        goto    b4
setb3
        bsf     SPG_stat,3

b4      btfsc   portc,4                  ; portc,4 = Norm sequence / CV- preset
        goto    setb4
        bcf     SPG_stat,4
        goto    b5
setb4
        bsf     SPG_stat,4
b5      btfsc   portc,5                 ; ext trigger positive flank / both flanks
        goto    setb5
        bcf     SPG_stat,5
        goto    b6
setb5
        bsf     SPG_stat,5
b6      btfsc   porta,2                 ; porta,2 = sequence start / stop
        goto    setb6
        bcf     SPG_stat,6
        goto    b7
setb6
        bsf     SPG_stat,6
b7      bcf     SPG_stat,7              ; switch modus
;       goto    sequence_init
;-------------------------------------------------
; Sequence init
;-------------------------------------------------
sequence_init
        lfsr    fsr1, 88h               ; Aantal presets waar doorheen gesequenced gaat worden
        movff   indf1, prestelu         ; Teller voor Up
        movff   indf1, presteld         ; teller voor down

        btfss   SPG_stat,3              ; interne of externe timing
        goto    int_seq_init            ; portc,3=0, intern

ext_seq_init                            ; portc,3=1, externe
        bcf     t0con,7                 ; timer0 uit
        bcf     intcon,5                ; interrupt disable tmr0
```

```
        bsf     intcon,4                ; INT0 interrup enable
        bcf     int_stat,0              ; int flag reset
        btfsc   SPG_stat,5              ; Beide falling and Rising? (ofwel INT1 ook aanzetten)?
        goto    both
        bcf     intcon,4                ; INT0IE disable
        bsf     intcon3,3               ; INT1IE (Interrupt enable)
        bsf     intcon2,6               ; Rising only
        goto    seq_start
both
        bsf     intcon,4                ; INT0IE (interrup enable)
        bsf     intcon3,3               ; INT1IE (Interrupt enable)
        bsf     intcon2,6               ; INTEDG0 (Rising)

        bcf     intcon2,5               ; INTEDG1 (Falling)
        bcf     int_stat,0              ; interrupt flag reset
        goto    seq_start

;-----------------------------------------
int_seq_init                            ; portc,3=0, intern
        bcf     intcon,4                ; INT0IE (interrupt disable)
        bcf     intcon3,3               ; INT1IE (interrupt disable)
        bsf     intcon,5                ; interrupt enable tmr0
        bcf     t0con,7                 ; timer even uit
        bcf     int_stat,0              ; int_flag reset
        bsf     t0con,7                 ; timer weer aan

;-----------------------------------------------------------------
; start sequence
;-----------------------------------------------------------------
seq_start
        btfsc   SPG_stat,6               ; nog steeds sequencen?
        goto    seq
        call    empty_stop
        goto    main
seq
        btfsc   SPG_stat,               ; CV to preset?
        goto    cv_preset               ; yes
        btfss   SPG_stat,1              ; No, up or down
        goto    seq_down
;-------------------------------------------------
; sequence UP

seq_up
        movlw   0x01                    ; up
        movwf   a_preset
        movff   presteld, downteller
seq_up_cont
        btfsc   int_stat,0              ; Interrupt al geweest? (kan timer OF trigger zijn)
        goto    activate_up
        lfsr    fsr1,8bh                ; status lezen
        movff   indf1, SPG_stat
        btfsc   int_stat,0
        goto    activate_up
        call    timerchoice
        btfsc   int_stat,0
        goto    activate_up
        btfss   SPG_stat,6              ; als interrupt niet komt (geen trigger)
        goto    seq_start               ; wel uit de loop springen
        goto    seq_up_cont
activate_up
        call    writepatch              ; schrijf de actieve preset
        call    trigger                 ; actief
        bcf     int_stat,0              ; interrupt flag reset (ext/int)
        incf    a_preset
        decfsz  downteller
        goto    seq_up_cont
        movlw   0x01                    ; ja, begin opnieuw
        movwf   a_preset
        movff   presteld, downteller
        goto    seq_start
```

```
;-------------------------------------------------
; Sequence DOWN
; a_preset is actieve teller
seq_down                                ; begin waarde laden
        movff presteld, a_preset
seq_down_cont
        btfsc   int_stat,0              ; Interrupt al geweest? (kan timer OF trigger zijn)
        goto    activate_down
        lfsr    fsr1,8bh                ; status lezen
        movff   indf1, SPG_stat
        btfsc   int_stat,0
        goto    activate_down
        call    timerchoice
        btfsc   int_stat,0
        goto    activate_down
        btfss   SPG_stat,6              ; als interrupt niet komt (geen trigger)
        goto    seq_start               ; wel uit de loop springen
        goto    seq_down_cont
activate_down
        call    writepatch              ; schrijf data
        call    trigger                 ; activeer
        call    timerchoice             ; test
        bcf     int_stat,0              ; reset int flag (int/ext)
        decfsz  a_preset
        goto    seq_down_cont
        movff   presteld, a_preset      ; opnieuw
        call    timerchoice
        goto    seq_start
; -------------------------------------------------
; CV to preset
; Analog value converted to preset
; adjust value to amount of presets in ram
cv_preset
        bcf     intcon,5                ; interrupt disable tmr0
        bcf     t0con,7                 ; timer uit
cvseq_main
        call    readcvforpreset
        call    cv2preset               ; read CV on An11 and convert number to a_preset
cvseq_wait
        lfsr    fsr1,8bh                ; status lezen
        movff   indf1, SPG_stat
        btfss   SPG_stat,6
        goto    seq_start
act_cv_pres
        call    writepatch              ; schrijf de actieve preset
        call    trigger                 ; actief
        goto    cvseq_main
;---------------------------------
; Convert Cv (AN11) to a_preset
; done by shifting the ad result
cv2preset
        lfsr    fsr1, 90h
        movff   indf1,tmppresh
        incf    fsr1l
        movff   indf1,tmppresl
        lfsr    fsr1, 88h
        movff   indf1, prescvtel
                                        ; depending on the amount of presets (N, or
                                        ; lfsr 88h) changing the scale of the CV to preset
        movlw   d'3'                    ; presetd = smaller then 3
        cpfsgt  prescvtel
        goto    pr3
        movlw   d'7'
        cpfsgt  prescvtel
        goto    pr7
        movlw   d'15'
        cpfsgt  prescvtel
        goto    pr15
        goto    pr32
pr3
        rrncf   tmppresh
        rrncf   tmppresh
        movlw   b'00000011'
        andwf   tmppresh
```

```
        movff   tmppresh, a_preset
        return
pr7
        rrncf   tmppresh
        movlw   b'00000111'
        andwf   tmppresh
        movff   tmppresh, a_preset
        return
pr15
        movlw   b'00001111'
        andwf   tmppresh
        movff   tmppresh, a_preset
        return
pr32
        rlncf   tmppresh                ; highbyte 1 bit to left
        btfss   tmppresl,7              ; lowbyte test bit 7
        goto    notset
        bsf     tmppresh,0
        goto    setpreset
notset
        bcf     tmppresh,0
setpreset
        movlw   b'00011111'             ; only b0-b4 are important
        andwf   tmppresh
        movff   tmppresh, a_preset
        return
; ---------------------------------------
; read ad convertor on input B4 (AN11) -
; cv to preset
readcvforpreset
        movlw   B'00101100'              ; channel An11 (portb,4) als input analog;
        movwf   ADCON0                  ; Ad staat nog even uit
        movlw   b'00000000'
        movwf   adcon1
        movlw   d'6'
        movwf   teller
aquis
        decfsz  teller
        goto    aquis
        bsf     adcon0,0                ; zet AD converter aan
        bsf     adcon0,1                ; set go/done bit
godo
        btfsc   adcon0,1
        goto    godo
        lfsr    fsr1, 90h               ; waarde voor (exp) cv to preset
        movff   adresh, postinc1
        movff   adresl, indf1           ; 91h
        movlw   b'00001110'             ; om poort b0-b3 weer te kunnen gebuiken digitaal
        movwf   adcon1                  ; adcon1 weer terugzetten naar deze waarde
        return
; ---------------------------------------
; read ad convertor on input A0
; cv to speed
Readcvforspeed
        movlw   B'00000000'             ; channel 0 An0  (porta,0) als input analog;
        movwf   ADCON0                  ; Ad staat nog even uit
                                        ; b6 en b7 nc
        movlw   b'00000000'
        movwf   adcon1
        movlw   d'6'
        movwf   teller
aquisition
        decfsz  teller
        goto    aquisition
        bsf     adcon0,0                ; zet AD converter aan
        bsf     adcon0,1                ; set go/done bit
godone
        btfsc   adcon0,1
        goto    godone
        lfsr    fsr1, 92h               ; waarde naar 92h
        movff   adresh, postinc1
        movff   adresl, indf1           ; 93h
        movlw   b'00001110'             ; om poort b0-b3 weer te kunnen gebuiken digitaal
        movwf   adcon1                  ; adcon1 weer terugzetten naar deze waarde
```

```
return
;-------------------------------------------------
; Leegschrijven matrix bij einde sequence
empty_stop
   btg    portb,7
          bcf      t0con,7              ; sequence stoppen en ad75019 leeg schrijven
          movlw    d'32'                ; 256 x 0 naar AD75019
          movwf    teller
s_end                                   ; AD75019 op nul zetten
          clrf     waarde               ; 32 x 8 bits sturen
          call     writedata
          decfsz   teller
          goto     s_end
          call     trigger              ; maak actief
          return
;-------------------------------------------------
; Timer choice. CV input of OSC lfsr 52h en 53h
; test of timer/cv speed determines timer:
; porta,5 =1, CV (porta,0) porta,5=0

timerchoice
          btfss    SPG_stat,2
          goto     osc_timer            ; no
cv_timer                                ; yes
          call     readcvforspeed
          lfsr     fsr1, 88h            ; vooralsnog waarden van osc
          movff    indf1, prestelu      ; counters
          movff    indf1, presteld      ; up and down
   lfsr   fsr1,92h                      ; timer waarde msb
          movff    indf1, adh
          lfsr     fsr1, 93h            ; timer waarde lsb
          movff    indf1, adl
          return
osc_timer                               ; waarde uit osc tabel halen
          lfsr     fsr1, 88h            ; ook de aantal preset waarde
          movff    indf1, prestelu
          movff    indf1, presteld
          lfsr     fsr1, 89h            ; timer waarde msb van /cf
          movff    indf1, adh
          lfsr     fsr1, 8ah            ; timer waarde lsb van /cf
          movff    indf1, adl
          return
;-----------------------------------------
; programmeer mode
prog
          btfss    recflag,1            ; extra test
          goto     main
          call     transfer
          bcf      recflag,0            ; all flags osc-interrupt reset
          bcf      recflag,1
          goto     main
;-------------------------------------------------
; manual mode. Check porta,1 (manual/On/Off)
; check portb2 (next)
; check portb3 (prev)
mlstart
          movlw    0x01                 ; begin waarde actieve preset
          movwf    a_preset
          call     writepatch           ; schrijf de actieve preset
          call     trigger              ; actief
          call     writeled
manloop
          btfsc    porta,1              ; manualmode?
          goto     manual
          call     empty_stop           ; nee, stoppen AD75019 leeg
          movlw    0x00
          movwf    a_preset
          call     writeled
          goto     main
manual
          btfss    portb,2              ; next
          goto     a_next
          btfss    portb,3              ; previous
          goto     a_prev
```

```
        goto    manloop
a_next
        call    waitxs                  ; wait
        btfsc   portb,2                 ; nogmaals schak testen
        goto    manloop
        incf    a_preset
        movlw   0x20
        cpfseq  a_preset
        goto    a_next2
        movlw   0x01
        movwf   a_preset
a_next2
        call    writepatch              ; schrijf de actieve preset
        call    trigger                 ; actief
        call    writeled
        goto    manloop
a_prev
        call    waitxs                  ; wait
        btfsc   portb,3                 ; nogmaals schak testen
        goto    manloop                 ; moet waarschijnlijk wachtloopje voor .. 100 ms?

        decfsz  a_preset
        goto    a_prev2
        movlw   0x20                    ; maximaal 25 presets
        movwf   a_preset
a_prev2
        call    writepatch              ; schrijf de actieve preset
        call    trigger                 ; actief
        call    writeled
        goto    manloop
;-----------------------------------------
; transfer received string
transfer
        lfsr    fsr1, 50h
        movff   indf1, preset_id        ; preset_id
trans1
        movlw   0x01
        cpfseq  preset_id               ; als gelijk naar transfer 1
        goto    trans2
        movlw   d'32'
        movwf   teller
        lfsr    fsr1, 54h
        lfsr    fsr2, 100h
t1      movff   postinc1, temp
        movff   temp, postinc2
        decfsz  teller
        goto    t1
        return
trans2
        movlw   0x02
        cpfseq  preset_id
        goto    trans3
        movlw   d'32'
        movwf   teller
        lfsr    fsr1, 54h
        lfsr    fsr2, 120h
t2      movff   postinc1, temp
        movff   temp, postinc2
        decfsz  teller
        goto    t2
        return
                                        … etc… (trans3 - trans31)
trans32
        movlw   0x20
        cpfseq  preset_id
        return
        movlw   d'32'
        movwf   teller
        lfsr    fsr1, 54h
        lfsr    fsr2, 4e0h
t32     movff   postinc1, temp
        movff   temp, postinc2
        decfsz  teller
        goto    t32
        return
```

```
;-----------------------------------------------
; WritePatch
writepatch
        btfss   porta,3
        goto    testcfbyte
        bsf     SPG_stat,0              ; op basis van dit byte wel/geen osc terug sturen
        goto    p1
testcfbyte
        btfsc   cf_byte,0
        goto    oscsetbit
        goto    oscclearbit
oscsetbit
        bsf     SPG_stat,0
        goto    p1
oscclearbit
        bcf     SPG_stat,0
p1      movlw   d'1'
        cpfseq  a_preset
        goto    p2
        call    wp1
        return
p2      movlw   d'2'
        cpfseq  a_preset
        goto    p3
        call    wp2
        return
        …                               … etc… (p3-p31)
p32     movlw   d'32'
        cpfseq  a_preset
        return
        call    wp31
        return
;----------------------------------------
; Active direct Patch (osc receive) 54-73
; data schrijven naar AD75019 in 8x8 opzet
; Wel alle 256 bits sturen. Laatste bit (x15,y15) eerst.
; Tijd: 580uS
wp0
        movlw   d'32'                   ; wp0 is not part of the sequence
        movwf   teller
        lfsr    fsr2, 73h
n0      movff   postdec2, waarde
        call    writedata
        decfsz  teller
        goto    n0
        btfss   SPG_stat,0
        return
        call    send_wp0
        return
;---------------------
; patch 1 (100h-11fh)
wp1
        movlw   d'32'
        movwf   teller
        lfsr    fsr2, 11fh
n1      movff   postdec2, waarde
        call    writedata
        decfsz  teller
        goto    n1
        btfss   SPG_stat,0
        return
        call    send_wp1
        return
        …                               …etc… (wp2-wp31)
;----------------------------------------
; patch 32 (4e0-4ff)
wp32
        movlw   d'32'
        movwf   teller
        lfsr    fsr2, 4ffh
n32     movff   postdec2, waarde
        call    writedata
        decfsz  teller
        goto    n32
```

```
        btfss   SPG_stat,0
        return
        call    send_wp32
        return
;-----------------------------------------------------
; subroutine write data aanroepen met 'waarde'
writedataportc,0                ; geen carry dus nul
        goto    write_1
write_0
        bsf     portc,0         ; wel carry, dus een een
write_1
        nop
        bsf     portc,1         ; shift Clk SHCP
        nop                     ; schuift op pos. flank
        bcf     portc,1
        decfsz  bitcnt
        goto    write_x
        return
;-----------------------------------
; Trigger (a2) klokken op timer
; Storage clock AD75019
trigger
        bsf     portc,2         ; Storage Clock
        nop
        bcf     portc,2
        nop
        bsf     portc,2
        return
;_____________________________________________________________________________________________
; Send routines
;_____________________________________________________________________________________________
; ------------------------------
; OSC send wp0
; /w1 ,iii iiii ii000 values (9 integers)
; ------------------------------
send_wp0
        call    sendslash
        movlw   a'w'
        movwf   txreg
        call    txwait
        movlw   a'0'            ; preset 0 is de active preset en maakt geen
        movwf   txreg           ; deel uit van de sequence
        call    txwait          ; vandaar een aparte header en niet op basis
        movlw   0x00            ; van a_preset (variabele)
        movwf   txreg
        call    txwait
        movlw   0x2c
        movwf   txreg
        call    txwait
        call    send_i
        call    send_control
        lfsr    fsr1, 54h
        movlw   d'32'
        movwf   teller
t0      movff   postinc1, txreg
        call    txwait
        decfsz  teller
        goto    t0
        call    send_oscend
        return
; ------------------------------
; OSC send wp1
; /01 ,iii iiii ii000 values (9 integers)
; ------------------------------
send_wp1
        call    sendslash
        movlw   a'0'
        movwf   txreg
        call    txwait
        movlw   a'1'
        movwf   txreg
        call    txwait
        call    send_nulcomma
        call    send_i
```

```
            call    send_control
            lfsr    fsr2, 100h
            movlw   d'32'
            movwf   teller
st1         movff   postinc2, txreg
            call    txwait
            decfsz  teller
            goto    st1
            call    send_oscend
            return
; ----------------------------
; OSC send 02
; /02 ,iii iiii ii000 values (9 integers)
; ----------------------------
send_wp2
            call    sendslash
            movlw   a'0'                    ;0
            movwf   txreg
            call    txwait
            movlw   a'2'                    ;2
            movwf   txreg
            call    txwait
            call    send_nulcomma
            call    send_i
            call    send_control
            lfsr    fsr2, 120h
            movlw   d'32'
            movwf   teller
st2         movff   postinc2, txreg
            call    txwait
            decfsz  teller
            goto    st2
            call    send_oscend
            return
            …                           …etc… (send_wp3 - send_p31)
; ----------------------------
; OSC send wp32
; /32 ,iii iiii ii000 values (9 integers)
; ----------------------------
send_wp32
    call    sendslash
            movlw   a'3'
            movwf   txreg
            call    txwait
            movlw   a'2'
            movwf   txreg
            call    txwait
            call    send_nulcomma
            call    send_i
            call    send_control
            lfsr    fsr2, 4e0h
            movlw   d'32'
            movwf   teller
st32
            movff   postinc2, txreg
            call    txwait
            decfsz  teller
            goto    st32
            call    send_oscend
            return
;-----------------------------------------------
; send de slash
sendslash
            movlw   a'/'
            movwf   txreg
            call    txwait
    return
```

```
;-----------------------------------------------
; send 1 x 0 en ','
send_nulcomma
        movlw   0x00
        movwf   txreg
        call    txwait
        movlw   0x2c
        movwf   txreg
        call    txwait
        return
;---------------------------------
; send 9 x i
send_i
        movlw   d'9'                    ; send 9 times 'i'
        movwf   teller                  ; en aanvullen met
i9      movlw   0x69                    ; 69 = ascii voor i
        movwf   txreg
        call    txwait
        decfsz  teller
        goto    i9
        movlw   0x00
        movwf   txreg
        call    txwait
        movlw   0x00
        movwf   txreg
        call    txwait
        return
;---------------------------------------
; send control byte
send_control
        lfsr    fsr1, 50h               ; inhoud contrle byte versturen
        movff   indf1, txreg
        call    txwait
        lfsr    fsr1, 51h
        movff   indf1, txreg
        call    txwait
        lfsr    fsr1, 52h
        movff   indf1, txreg
        call    txwait
        lfsr    fsr1, 53h
        movff   indf1, txreg
        call    txwait
        return
;----------------------------------------
; osc-end byte send
send_oscend
        movlw   0x00                    ; aanvullen met 2 x 00
        movwf   txreg
        call    txwait
        movlw   0x00
        movwf   txreg
        call    txwait
        movlw   0xff                    ; send char FF en FF
        movwf   txreg
        call    txwait
        movlw   0xff
        movwf   txreg
        call    txwait
        return
;-----------------------------------------------
; /cf retour
; ----------------------------------------------
retour_config
        movlw   a'/'                    ; /cd0
        movwf   txreg
        call     txwait
        movlw    a'c'
        movwf   txreg
        call    txwait
        movlw    a'd'
        movwf   txreg
         call    txwait
        movlw    0x00
        movwf    txreg
        call    txwait
```

```
        movlw   a','
        movwf   txreg
        call    txwait
        movlw   a'i'
        movwf   txreg
        call    txwait
        movlw   0x00
        movwf   txreg
        call    txwait
        movlw   0x00
        movwf   txreg
        call    txwait
        lfsr    fsr2, 88h               ;89 (timer high) /cf byte 2
        movff   postinc2, txreg
        call    txwait
        movff   postinc2, txreg
        call    txwait
        movff   postinc2, txreg         ;8a (timer low) / cf byte 1
        call    txwait
        movff   postinc2, txreg         ;8b (timer low) / cf byte 1
        call    txwait
        return
; ------------------------------------------------
; send routine RS232
;------------------------------------------------
txwait
        btfss   pir1,4                  ; check of zendregister leeg is (1= register leeg)
        goto    txwait                  ; als register 0 (vol) is
        return
;------------------------------------------------
; Write active preset id to 5 x LED
; portb,5 data / portb,7 clock / portb,6 shift clock
;------------------------------------------------
writeled                                ; write 5 x leds
        movff   a_preset, tmpdisp
        movlw   d'8'
        movwf   ledcnt
wled_x
        rlcf    tmpdisp
        bc      wled_0                  ; branch if carry
        bcf     portb,5                 ; geen carry dus nul
        goto    wled_1
wled_0
        bsf     portb,5                 ; wel carry, dus een 1
wled_1
        nop
        bsf     portb,7                 ; Clock
        nop                             ; schuift op pos. flank
        bcf     portb,7
        decfsz  ledcnt
        goto    wled_x
        bsf     portb,6                 ; shiftclock
        nop
        bcf     portb,6
        return
; ------------------------------------------------
waitxs
        movlw   0xff                    ; main counter v1
        movwf   var1
        movlw   0xff                    ; second couner v2
        movwf   var2                    ; time is V1 x V2 x V3 instruction cycle
        movlw   0x04                    ; delay is nu ongeveer 100mS
        movwf   var3
dl1     decfsz  var1
        goto    dl1
        movlw   0xff
        movwf   var1
dl2     decfsz  var2
        goto    dl1
        movlw   0xff
        movwf   var2
dl3     decfsz  var3
        goto    dl2
        return
        end
```

Technical specifications SPG (Linked to Assembly version 85)

Powersupply The SPG needs +15V (120mA), -15V(120mA), +5V(250mA) and GND. The power supply should be part of the system and I used the 'Traco-Power TML 15515' for the power-supply. The euro-rack version (SPG model 6) can be connected to the power supply of the Doepfer A-100 system.

Inputs and outputs The matrix, the AD75019 from Analog Devices, can switch audio signals between -12V and +12V. Higher values will cause distortion. The opamps (NE5532 and TL072) are working on the same power-rail from -12V to +12V.

Control voltages general: In the inside of the SPG all digital signals are +5V or 0V. Also the PIC18F2523 microcontroller with its Analog to Digital convertor (ADC) can only handle 0-5V. Since there are a lot of different standars regarding the control voltages and modulair synth's, the SPG models have some modifications on board to re-scale these values to the appropriate values.

CV input model 2, model 3 and model 6 (euro-rack): These inputs can take -5V to +5V on the CV input. This is the audio standard from Doepfer (A-100 set). The value will be re-scaled internally from 0 to 5V.

Trigger input model 2, model 3 and model 6 (euro-rack): This input can handle 0-12V. The value will be re-scaled from 0 to 5V.

Max/Msp patch. The Max/Msp patch can be downloaded from http://www.ipson.nl. The patch is created with Max/Msp version 7.1

Bibliography

(1999, Rev C). Analog Devices AD75019 Datasheet.

Erica synths. Retrieved 1-5-2017, from http://www.ericasynths.lv/en/shop/eurorack-modules/by-functions/sequencers/matrix-mixer/

(2014). Lantronics Xport Embedded Device Server Data sheet. (910-8151)

(2006). Microchip PIC18F2325 Datasheet. 390.

Chamberlin, H. & Chamberlin, H. (1980). *Musical Applications Of Microprocessors*. RochellePark,N.J.: Hayden Book Co.

Bryan Mayton, Gershon Dublon, Nicholas Joliat (2012). Patchwerk: Multi-User Network Control of a Massive Modualr Synthesizer.

F. van Herwijnen, H Leegwater (1997). *Poly-Technisch zakboekje*. (48e druk ed.) Arnhem: Koninklijke PBNA.

A.J. van den Broek (2010 - 2017). *Technology for Art and Education*. Retrieved 21-3-2017, from http://www.ipson.nl

Horowitz, P. & Hill, W. (1989). *The Art of Electronics*. (2nd ed. ed.) Cambridge: Cambridge University Press.

Endnotes

[1] Kees Tazelaar (1962) followed courses in Sonology in Utrecht and The Hague, and later studied composition under Jan Boerman at the Royal Conservatoire. He has been teaching at the Institute of Sonology since 1993 and has been head of the institute since 2006. His electronic music features a combination of formalisation, richness of sound and a compositional approach to sound spatialisation (sonology.org)
[2] Jo Scherpenisse (1938) worked as a head of the Electronics workshop and as technician for the Institute of Sonology from 1965-2003.
[3] MIDI: Musical Instrument Digital Interface, is a technical standard that describes a protocol or digital interface and allows a wide variety of musical instruments and/or computers to connect and communicate with one another (Wiki).
[4] Open Sound Control (OSC) is a protocol for communication among computers, sound synthesizers, and other multimedia devices that are optimized for modern networking technology. Bringing the benefits of modern networking technology to the world of electronic musical instruments, OSC's advantages include interoperability, accuracy, flexibility, and enhanced organization and documentation (Wiki).
[5] Karlheinz Stockhausen Studio is the main studio of the Composition department of the Royal Conservatoire and a workshop for composers.
[6] WMD is a manufacturer of music hardware including Modular Synthesizers, Guitar Pedals, and stand alone music instruments. https://www.wmdevices.com/products/sequential-switch-matrix
[7] RS-232 is a standard serial data communication protocol used amongst computers and peripherals.
[8] Microchip datasheet, or manual. Microchip technology 2006 DS39755A
[9] CMOS. Complementary metal–oxide–semiconductor, abbreviated as CMOS is a technology for constructing integrated circuits (Wiki)
[10] Reduced Instruction Set Controller (RISC). The general concept is that of a computer that has a small set of simple and general instructions, rather than a large set of complex and specialized instructions (Wiki).
[11] Eusart : Enhanced Universal Asynchronous Receiver / Transmitter. It is a type of a serial interface device that can be programmed to communicate asynchronously or synchronously (Wiki).
[12] Operational Amplifier, or Opamp is a DC coupled high-gain electronic voltage amplifier with a differential input and, usually, a single ended output (Wiki)
[13] A Spectrum Analyzer measures the magnitude of an input signal versus frequency within the full frequency range of the instrument. The primary use is to measure the power of the spectrum of known and unknown signals (Wiki).
[14] ASCII abbreviated from American Standard Code for Information Interchange, is a character encoding standard. It represent text in computers, telecommunication equipment, and other devices.
[15] Low Latency (LoLa) software. A Low Latency audio visual streaming system developed by the 'Conservatorio di musica Giuseppe Tartini, in Triest.
[16] http:/www.ipson.nl. Internet Protocol Sonology. Technology for Art and Education, created and maintained by Lex van den Broek.