EINDHOVEN UNIVERSITY OF TECHNOLOGY

Eindhoven University of Technology

MASTER

Fabrication of complex nanowire structures for topological quantum computing

Schellingerhout, A.G. (Sander)

Award date:
2019

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Department of Applied Physics
Advanced Nanomaterials and Devices

Fabrication of complex nanowire structures for topological quantum computing

Master Thesis

Sander Schellingerhout

Supervisors:

Main Supervisor	Prof.Dr.	Erik Bakkers
Daily Supervisor	MSc.	Saša Gazibegović
Daily Supervisor	MSc.	Roy op het Veld
Daily Supervisor	MSc.	Ghada Badawy

December 7, 2018

Abstract

Quantum computing is a promising development for many fields as it can exponentially speed up certain calculations. In topological quantum computing a very stable and scalable system is predicted, which relies on one-dimensional semiconductor structures. This thesis outlines an effort to fabricate such semiconductor structures suitable for a single qubit device. This is done on a platform that allows single nanowires to merge into networks, with the goal of 3x3 or larger networks. This is first attempted using pure InSb nanowires. This was unsuccessful, after which the more conventional method of growing InSb nanowires on top of InP nanowires is used successfully to fabricate nanowire networks suitable for a single qubit device.

Contents

1 **Introduction** 4

2 **Theory** 5
- 2.1 Quantum computing . . . 5
- 2.2 Topological quantum computing . . . 7
  - 2.2.1 The basics . . . 7
  - 2.2.2 First signatures of the Majorana Zero Mode . . . 10
  - 2.2.3 Braiding and coherent transport . . . 12
  - 2.2.4 The single qubit and beyond . . . 14
- 2.3 Nanowires and structures . . . 15
  - 2.3.1 Nanowires . . . 15
  - 2.3.2 Vapour-Liquid-Solid growth . . . 16
  - 2.3.3 Crystal structure . . . 17
  - 2.3.4 Nanowire networks . . . 18

3 **Experimental methods** 20
- 3.1 Fabrication tools . . . 20
  - 3.1.1 Plasma Enhanced Chemical Vapor Deposition (PECVD) . . . 21
  - 3.1.2 Resist spinning, baking and developing . . . 22
  - 3.1.3 Electron beam lithography (EBL) . . . 24
  - 3.1.4 Reactive Ion Etching (RIE) . . . 26
  - 3.1.5 Electron beam physical vapor deposition . . . 27
  - 3.1.6 Metalorganic vapor phase epitaxy (MOVPE) . . . 28
  - 3.1.7 Scanning electron microscopy (SEM) . . . 31
- 3.2 Sample preparation . . . 32
  - 3.2.1 Design . . . 32
  - 3.2.2 Phase one: Markers . . . 35
  - 3.2.3 Phase two: Trenches . . . 36
  - 3.2.4 Phase three: Dots . . . 38

4 Results **39**
4.1 Growth recipes 40
4.1.1 InP nanowires 40
4.1.2 InSb nanowires 41
4.2 Stemless InSb nanowires 42
4.3 InSb nanowires on InP stems 45
4.3.1 InP stem length and encapsulation 45
4.3.2 Three step InSb growth 53
4.3.3 Optimizing InSb length 55
4.3.4 Merging behaviour 58

5 Conclusion **61**
5.0.1 Outlook 62

A Growth experiments **66**

B Design software code **67**

Chapter 1

Introduction

Since the discovery of the transistor in 1947 the device revolutionized the field of electronics. Companies manage to cram more and more transistors on a silicon chip, leading to an exponential growth of computing power on a chip as described by Moore's law [1]. However, transistors cannot continue shrinking in size indefinitely, as the physical limits are being approached. Obviously transistor gates cannot become thinner than a single atom, and even for thicker layers electron tunnelling through the gate can destroy the transistor performance [2]. More and more elaborate fabrication methods and tools are needed to approach this fundamental limit, increasing the cost and time needed for the fabrication of higher density chips.

Fortunately a new star is rising in the world of computing. The quantum computer, of which pioneering work originates in 1979 by Paul Benioff and in 1982 by Richard Feynmann [3, 4], has been attracting huge amounts of attention. Corporations including Google, IBM and Microsoft are in a race to achieve quantum supremacy, which is the potential of a quantum computer to solve problems that a regular computer cannot feasibly do [5]. Currently the main challenges in quantum computing is simply the question of how to build one. This thesis outlines an effort to construct a fundamental building block for a specific type of quantum computer, the topological quantum computer.

Chapter 2

Theory

2.1 Quantum computing

A conventional computer uses transistors which can be turned on or off, representing either $|0\rangle$ or $|1\rangle$. A quantum computer employs the principles of quantum physics to create a system where $|0\rangle$ and $|1\rangle$ are represented by two states of a quantum mechanical system. These two-level quantum states, often referred to as quantum bits or qubits, can be put in what is called a superposition state. This superposition can represent not only $|0\rangle$ or $|1\rangle$, but also anything in between. This superposition state can be combined with another quantum mechanical effect, entanglement, in which two entangled particles are affected by each others state and operations on it, even if the particles are spatially separated. When employed on qubits the amount of entangled space grows exponentially with the amount of qubits, in turn exponentially growing the computational power of the system.

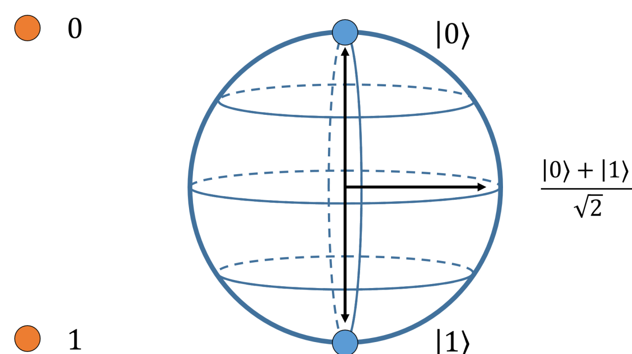


Figure 2.1: Representation of a regular bit on the left and a qubit (Bloch sphere) on the right.

The quantum computer as a principle is of interest to many fields. A popular example is the field of cryptography, in which a quantum computer could efficiently perform integer factorization using Shor's algorithm, which is impossible on a classical computer. Efficient integer factorization would effectively break many encryption schemes currently in use, which is likely the reason many governments are interested in quantum computing.

The most useful application discovered so far is the ability to efficiently simulate quantum mechanical systems. A many particle system is described by an exponentially growing Hilbert space, which requires exponential time on a classical computer. A quantum computer however could simulate a many particle system with an amount of qubits similar to the number of particles in the system [6]. This is of great interest to for example studying the folding of proteins, which are highly complex molecules essential for a large range of processes in the human body. A similar problem is the discovery of new drugs, where currently simulations are limited to small molecules. The ability to simulate large molecules using a quantum computer can help find new drugs, as well as save years of development time.

Most current approaches to building a two-level system suitable as a qubit involve either the spin-up and spin-down state of electrons or the quantized energy levels of an atom. This approach has a major downside: Quantum decoherence, in which the quantum state in the two-level system is collapsed by outside interference. A limited coherence time might not be an issue for a few qubit system, but for a quantum computer consisting of large amounts of qubits it can introduce a large amount of errors. Fortunately another approach exists: The topological qubit.

2.2 Topological quantum computing

2.2.1 The basics

A topological quantum computer is fundamentally different from conventional quantum computers in the sense that it employs one-dimensional quasi-particles instead of trapped quantum particles. The proposed particles to be used in a topological quantum computer are called Majorana fermions, first decribed by Ettore Majorana as a solution to the Dirac equation in 1937 [7]. Majorana fermions have been predicted to appear as quasiparticles in certain solid-state systems [8]. The property that makes these Majorana fermions so interesting is their non-Abelian exchange statistics. For regular bosons and fermions the following statistics apply: The position of two indistinguishable particles can be exchanged without changing the ground state of the system. Non-Abelian anyons however act differently. When the position of two Majorana particles is exchanged the system moves from one degenerate ground state to another degenerate ground state. This system can be used to create a quantum bit, which is predicted to be very stable due to the information not being encoded in a single particle but in two spatially separated quasiparticles [8].

The requirements for a system to make Majorana quasiparticles appear are far from trivial. In 2001 Alexei Kitaev proposed that these quasiparticles can appear at the edges of a spinless p-wave superconductor [9]. Unfortunately p-wave superconductors do not exist in nature, but a recipe to achieve Majoranas by inducing p-wave superconductivity was reported in 2010 [10]. Four key ingredients were identified: A one-dimensional semiconductor nanowire, induced p-wave superconductivity by deposition of an s-wave superconductor on top of the nanowire, strong spin-orbit coupling and an induced magnetic field.

A device that combines all these ingredients can be seen in Figure 2.2. A p-wave-like gap is induced in a 1D nanowire by putting a regular s-wave superconductor (Aluminium) in contact with the nanowire. The system is made effectively spinless by applying a magnentic field along the nanowire, which induces Zeeman splitting in the nanowire. Now by tuning the semiconductors chemical potential in between the split bands and thus forcing all electrons into the same spin state, the requirements for the appearance of Majorana quasiparticles are met.

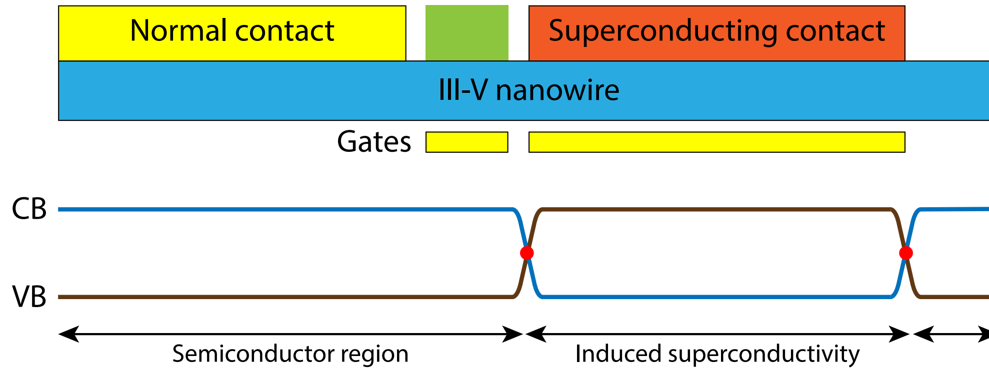


Figure 2.2: (Top) Schematic of a device which can be used to detect Majorana zero modes. The green rectangle indicates the tunnel barrier seperating the semiconducting (normal) part of the wire from the part with induced superconductivity. (Bottom) Schematic illustrating the energy states along the nanowire. The green line indicates the valence band (VB) and the blue line indicates the conduction band (CB). The red dot represents the Majorana quasiparticles.

In Figure 2.3 the electron energy dispersion in a nanowire is shown. Figure 2.3a shows the dispersion relation with no spin-orbit coupling and Zeeman splitting. Figure 2.3b includes the effect of spin-orbit coupling, which splits the spin-up and spin-down dispersion. Figure 2.3c includes both spin-orbit coupling and Zeeman splitting, where the Zeeman splitting opens up a gap between the two bands. The chemical potential in the nanowire device needs to be tuned inside this gap, which means the Zeeman splitting should ideally be as large as possible. This has to be done without destroying the superconductivity with a strong magnetic field, so it is important to use a material with an as large as possible Landé g-factor. Strong spin-orbit coupling is also desired, since stronger spin-orbit coupling means the electrons align more strongly to the magnetic field, and since otherwise the magnetic field suppresses the gap at finite momentum [11]. Materials that posses a large Landé g-factor and strong spin-orbit coupling are, for example, InSb and InAs.

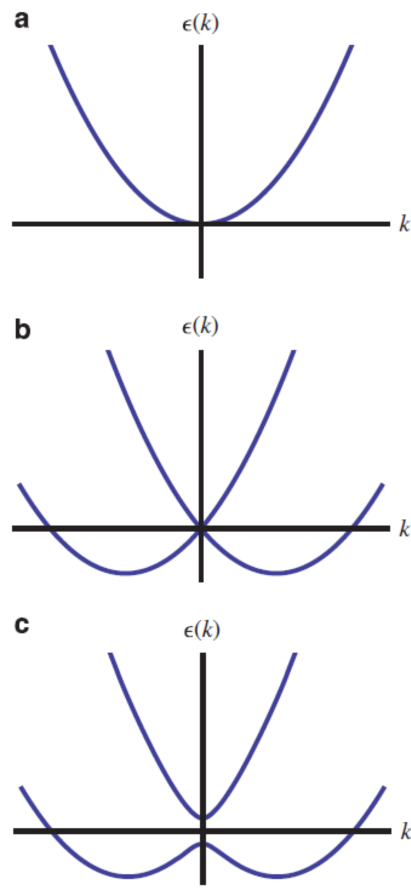


Figure 2.3: Electron energy as function of momentum for a 1D wire for (a) no spin-orbit coupling and Zeeman splitting (b) non-zero spin-orbit coupling and no Zeeman splitting (c) non-zero spin-orbit and Zeeman splitting [8].

2.2.2 First signatures of the Majorana Zero Mode

One of the first signatures of the existence of Majorana quasiparticles was reported in 2012 [12]. An InSb nanowire device as illustrated in Figure 2.2 was investigated. One superconducting contact and one normal contact are deposited on the nanowire, with a back gate for the superconducting region to tune the chemical potential and a back gate to create a tunnel barrier. A magnetic field is applied parallel to the nanowire. The Majorana quasiparticles are expected to appear at the edges of the superconducting contact, as illustrated by the red dots. These particles live in a zero-energy (midgap) bound state, often referred to as a Majorana zero mode (MZM), and should allow a tunneling current to flow at zero applied bias, as illustrated in Figure 2.4.

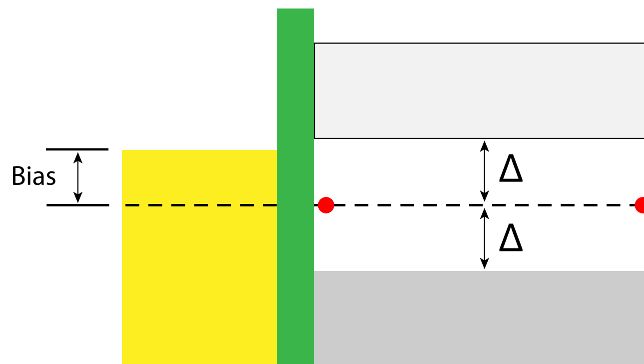


Figure 2.4: Illustration of the energy states. Green indicates the tunnel barrier separating the normal semiconducting nanowire part on the left and the nanowire part with the induced superconducting gap Δ. Red indicates the Majorana particles. Adapted from [12].

In Figure 2.5 the measured differential conductance of this device is shown. With no applied magnetic field no zero bias peak is observed, as expected from the requirement of Zeeman splitting for the Majorana quasiparticles to appear. With non-zero applied magnetic field a clear differential conductance peak at zero bias is observed, which is suspected to be caused by the Majorana zero mode. The theoretically predicted differential conduction of the MZM is $2e^2/h$, which can be seen in the line cut. The observation of quantized conductance at zero bias in this device strongly supports the existance of Majorana zero modes, which paves the way for future braiding experiments [13].

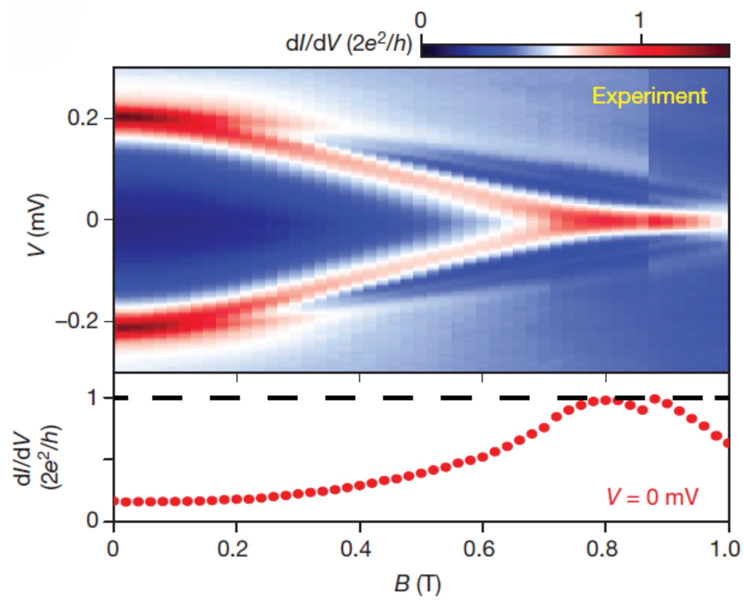


Figure 2.5: Differential conductance color plot showing the conductance peak at zero bias. The line cut is taken at zero bias showing that the conductance reaches the universal value of $2e^2/h$ [13].

2.2.3 Braiding and coherent transport

With the appearance of a MZM in a single InSb nanowire, the next logical step is the experiment of exchanging the position of the Majorana fermions as explained in section 2.2.1. This process is often referred to as braiding, and in Figure 2.6 an example of braiding MZMs at a T-junction is shown. The T-junction consists of a superconducting nanowire structure, with a locally tunable chemical potential. The solid lines indicate the topological regime, and by moving the domain walls using the tunable chemical potential the MZMs can be exchanged.

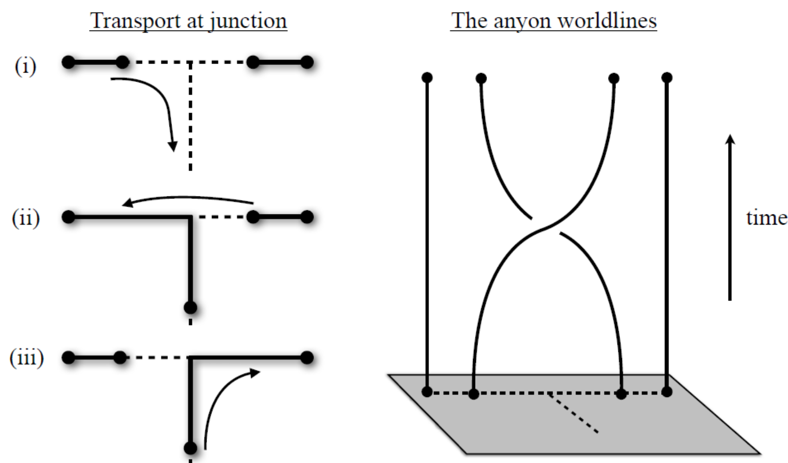


Figure 2.6: Example of the braiding of Majorana fermions in a T-junction superconducting nanowire. The chemical potential is locally tunable, allowing the domain walls of the topological regime, indicated with the solid lines, to be moved. [14]

As Majorana fermions are their own antiparticle, exchanging or braiding two particles cannot be done on a single nanowire as they will annihilate when brought close together. Many proposals exist for suitable structures which all employ single crystalline nanowire networks as key component for the braiding experiment. Proposed structures include a T-shaped nanowire structure, illustrated in Figure 2.6 [15], and a square nanowire structure as illustrated in Figure 2.7. One of the requirements for braiding experiments is long-distance coherence in the device. A proposed way to probe long-distance coherence is to look for Aharanov-Bohm oscillations in a device shown in Figure 2.7 [16].

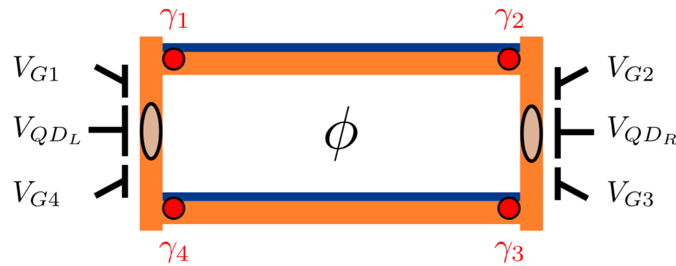


Figure 2.7: Experimental test for long-distance coherent transport through a 1D topological superconductor. If single electron transport is coherent Aharanov-Bohm oscillations should be seen when changing the enclosed flux ϕ. Orange indicates the semiconductor NW, blue indicates the superconductor. The red dots indicate the expected location of the MZM. Note that the Aharonov-Bohm measurement itself does not require superconductivity [16].

Such a device was fabricated and measured as shown in Figure 2.8 in collaboration between this group in Eindhoven and the TU Delft [17]. The device consists of four connected InSb nanowires with two normal contacts. Since the Aharonov-Bohm measurements do not require superconductivity, no superconductor is deposited. The Figure shows the nanowire device as well as the magnetoconductance showing periodic Aharonov-Bohm oscillations. This is proof that the grown nanostructures are of high crystalline quality with phase coherent transport, which brings us one step closer to performing actual braiding experiments.

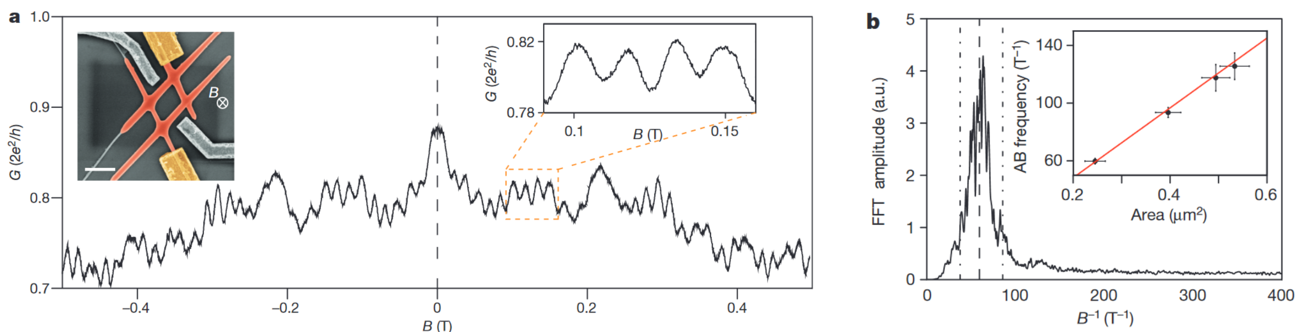


Figure 2.8: (a) Magnetoconductance measurement of the device shown in the top left corner which shows periodic Aharonov-Bohm oscillations. (b) FFT spectrum of the magnetoconductance indicating the Aharonov-Bohm oscillation frequency. The dashed line shows the expected frequency based on calculations of the area inside the device which match with the measurements [17].

2.2.4 The single qubit and beyond

With strong evidence for Majorana zero modes and coherent transport in 1D nanowire devices the obvious next step is the realisation of a single qubit. A design for a single and a two qubit device was reported by [18]. The qubit consists of a semiconductor 1D structure, with induced superconductivity at the blue areas and no superconductivity at the green areas. The orange area indicates a superconducting bridge, and the red region indicates quantum dots. The Majorana zero modes are indicated by the black cross.

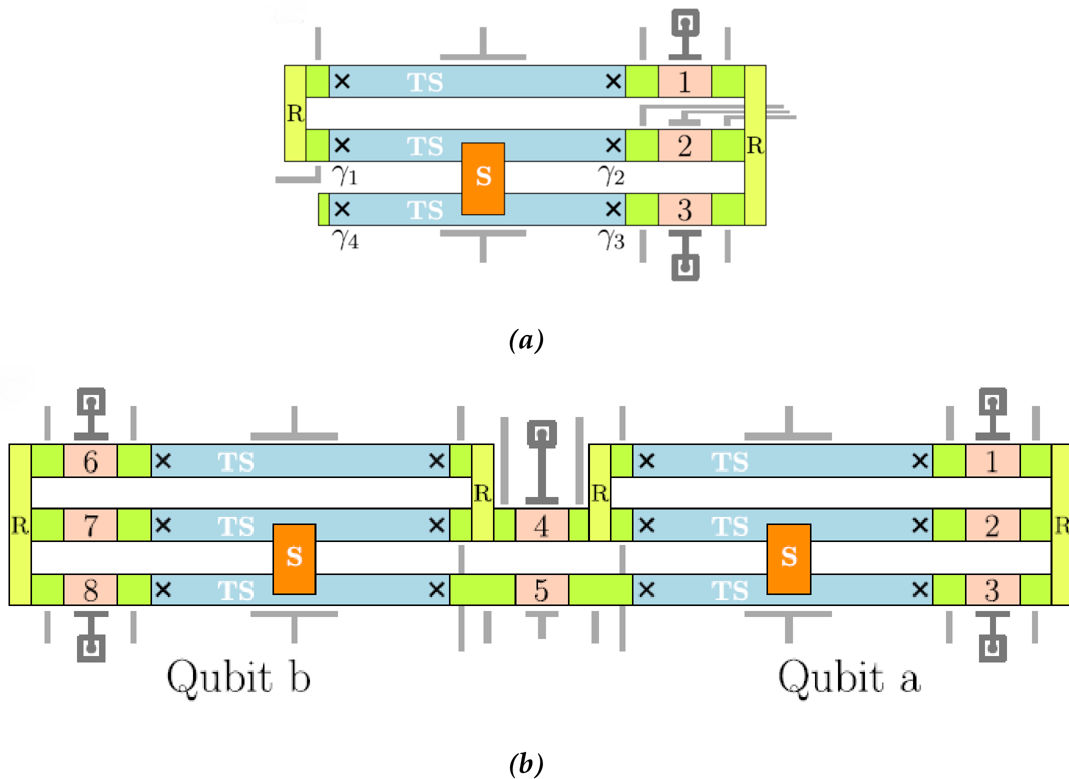


Figure 2.9: (a) Single qubit device. (b) Two qubit device. The green area indicates the semiconductor. The blue area indicates the semiconductor with induced superconductivity, with the black crosses indicating the location of the Majoranas. The orange area indicates a superconducting bridge and the red region indicates quantum dots used for controlling the device [18].

2.3 Nanowires and structures

2.3.1 Nanowires

III-V semiconductor NWs are crystalline pillar shaped objects. They are characterised by a large aspect ratio, with a diameter up to a few hundred nanometer and a length up to a few micrometer. They consist of materials from group III (e.g. Indium, Gallium) and V (e.g. Arsenic, Antimony) of the periodic table. They have been investigated for a large range of applications, including transistors, LEDs and solar cells [19–21].

Due to the sometimes large lattice mismatch in III-V semiconductors it is often impossible to fabricate defect free interfaces in bulk. However, nanowires have the remarkable ability to relieve strain induced by lattice mismatch due to their small diameter. When sufficiently small the diameter also helps confine the electron wave function, leading to a one-dimensional system. As explained before, the ingredients required for the appearance of Majorana zero modes are a 1D system (the nanowire), a large Landé g-factor, strong spin-orbit coupling and a long coherence length. The last three requirements are featured in the InSb material system, which is why this material is used in this work. InSb is often grown on top of an InP stem to mediate the growth [17].

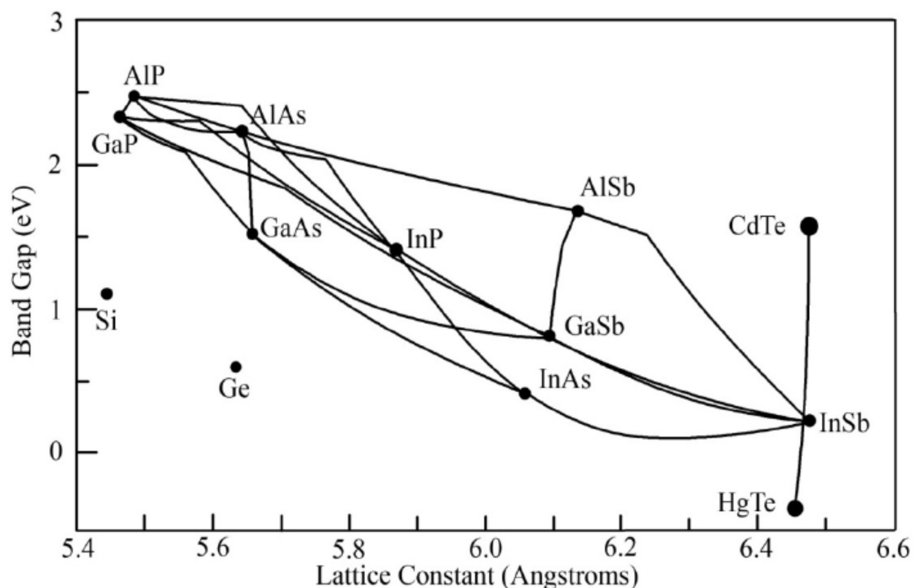


Figure 2.10: Lattice constant plotted against band gap for common semiconductor materials. [22]

2.3.2 Vapour-Liquid-Solid growth

Epitaxial nanowire growth is achieved by means of gold assisted (VLS) growth in an MOVPE reactor. The reactor itself will be explained more in section 3.1, here the general growth mechanism will be explained.

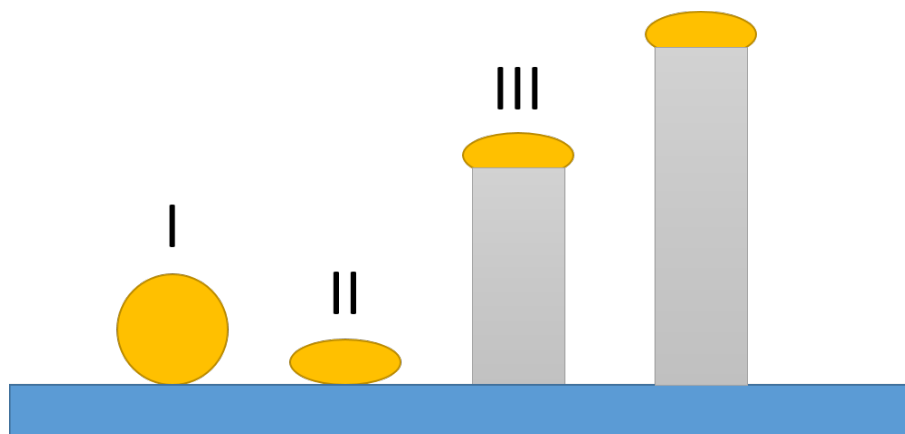


Figure 2.11: Schematic of VLS growth. I. A gold catalyst particle is deposited on the substrate. II. The particle is supersaturated in the reactor by constant supply of the gas phase material. III. Material nucleates underneath the particle leading to epitaxial growth of a crystalline nanowire.

The vapour-liquid-solid (VLS) growth technique was first reported by Wagner and Ellis in 1964 [23]. The VLS growth mechanism is often used for nanowire growth, as it is much faster and much more reliable than direct adsorption of a gas phase material on a substrate. As the name suggests, the III-V particles undergo three distinct phases as illustrated in Figure 2.11. First the gas phase precursors decompose and are absorbed into the liquid gold catalyst particle. After supersaturation in the gold catalyst is reached and nucleation starts the III-V particles are moved into their final solid phase. The layer by layer epitaxial growth occurs at the bottom of the catalyst droplet, lifting the droplet up and allowing the nanowires to continue to grow. In practise many parameters affect the way the nanowires grow. A sufficient temperature window, V/III ratio and total flow is needed, as well as a properly prepared substrate.

2.3.3 Crystal structure

The two most common crystal structures for III-V semiconductors are zinc-blende and wurtzite, which are illustrated in Figure 2.12.

(a) Zincblende.

(b) Wurtzite.

Figure 2.12: Stacking sequence of the crystal lattice for wurtzite and zincblende.

Both types of crystal have close packed layers, with the only difference the stacking sequence. Zinc-blende is characterized by ABCABC... stacking which results in the cubic close packed structures while wurtzite is characterized by ABAB... stacking which results in hexagonal close packed structure. The crystal structure most III-V semiconductor nanowires prefer, including InSb and InP, is zincblende. However, where InSb is usually grown defect free, InP is defect rich, with mixing zinc-blende and wurtzite parts, as well as stacking faults and rotational twins. This can have consequences for the InSb nanowires that are grown epitaxially on top of the InP stem, as there are two possible zinc-blende orientations which are rotated 180 degrees with respect to each other. The preferred growth direction for zinc-blende nanowires is the <111> direction, while wurtzite nanowires prefer to grow in the <0001> direction [24].

2.3.4 Nanowire networks

As explained in section 2.2.4 and 2.3.1, complex 1D networks are needed with the prime candidate InSb nanowires. Nanowires grow bottom-up, so an intricate growth scheme with crossing nanowires is required to make a device such as the single or two-qubit devices in Figure 2.9a and 2.9b.

First the right substrate has to be selected. The difficulty here comes from the fact that the nanowires need to cross each other. The nanowires prefer growth in the (111) direction, which rules out (111)B substrates as the nanowires will grow with a 90° angle towards the substrate. The easiest solution seems to be using a (100) substrate, on which the nanowires will grow at a 35.3° angle to the substrate [25]. The problem is that the nanowires can grow in two possible (111)B directions, making it a game of chance to fabricate complex structures. This is not a big issue for small networks such as nanocrosses (1x1 network), but the bigger the network, the lower the chance to successfully fabricate it. The solution arises in using a (100) substrate in which trenches are etched which have (111)B facets, as sketched in Figure 2.13. Now the nanowires can grow under a 90° angle to the (111)B facets, ensuring the nanowires will cross each other in the intended way.

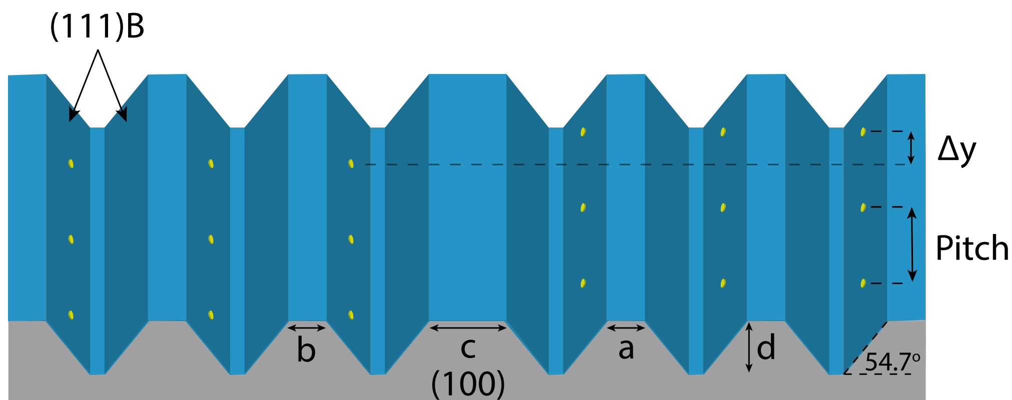


Figure 2.13: Schematic of trenches etched into a (100) substrate. Light blue represents (100) facets and dark blue represents (111)B facets. The yellow dots indicate the gold droplets used for VLS growth. a, b and c indicate the distance between the trenches, which determine how far the merging points of the structures are apart. d indicates the depth of the trenches. The pitch is the distance between two subsequent structures, and Δy indicates the offset in the structure to achieve merging without the nanowires getting fully blocked by one another.

Using this scheme single and two qubit devices such as in Figure 2.9a and 2.9b can be fabricated. In Figure 2.14 the equivalent bottom-up nanowire structures are shown.

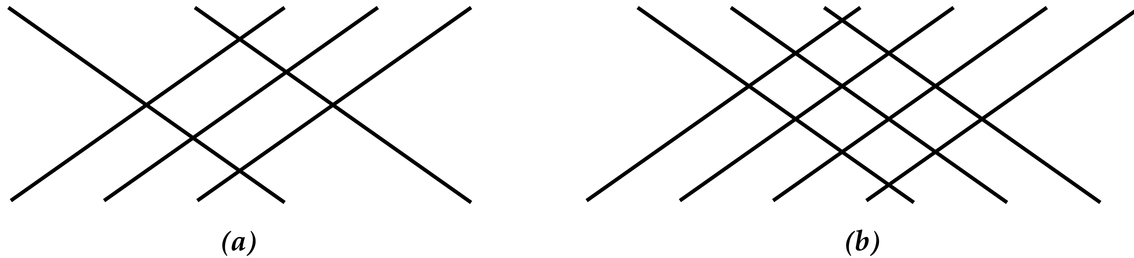


Figure 2.14: (a) Nanowire structure suitable for a single qubit device. (b) Nanowire structure suitable for a two qubit device.

Similar substrates are previously used to fabricate 1x1 nanocrosses and 2x2 nanohashtags [17], but larger structures were not achieved yet due to the evaporation of the InP stems on which the InSb nanowires are grown. Then the main research question of this thesis appears: How can nanowire structures suitable for a single qubit device be fabricated in a controlled and reproducible way?

The remainder of this thesis is structured as follows:

- **Chapter 3** explains the fabrication tools used for sample preparation as well as the nanowire growth method, metal-organic vapour phase epitaxy (MOVPE). Then the main analysis tool, scanning electron microscopy (SEM) is explained. The design software is explained and the fabrication process is summarized.
- **Chapter 4** describes and discusses the results obtained from growing both stemless InSb nanowires and InSb nanowires on InP stems in order to grow nanowire networks.
- **Chapter 5** presents the conclusion of the experiments as well as an outlook on further experiments.

Chapter 3

Experimental methods

3.1 Fabrication tools

In this section the experimental techniques for fabrication and characterisation are presented. First the sample preparation tools are briefly explained, after which the technique and reactor for nanowire growth, Metal-organic Vapour Phase Epitaxy (MOVPE), is explained. Finally the main characterisation tool, the Scanning Electron Microscope (SEM) is explained.

3.1.1 Plasma Enhanced Chemical Vapor Deposition (PECVD)

PECVD is a technique in which uniform and high quality thin-films can be deposited. This can be done at lower temperatures than that of standard Chemical Vapor Deposition (CVD), as the precursor decomposition is assisted by the generated RF plasma. In the process used for this thesis the substrate is heated to 300°C, compared to the 600°C to 800°C required for CVD which would damage the InP substrate.

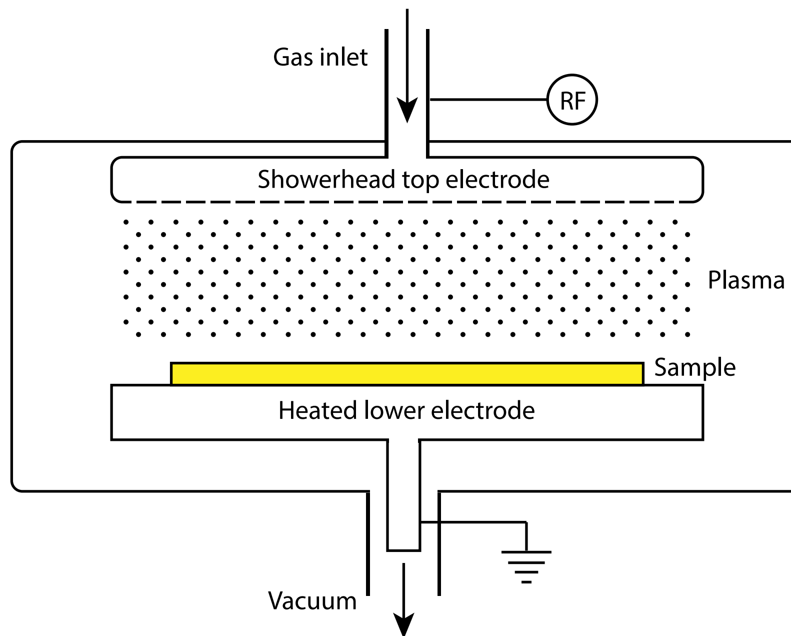


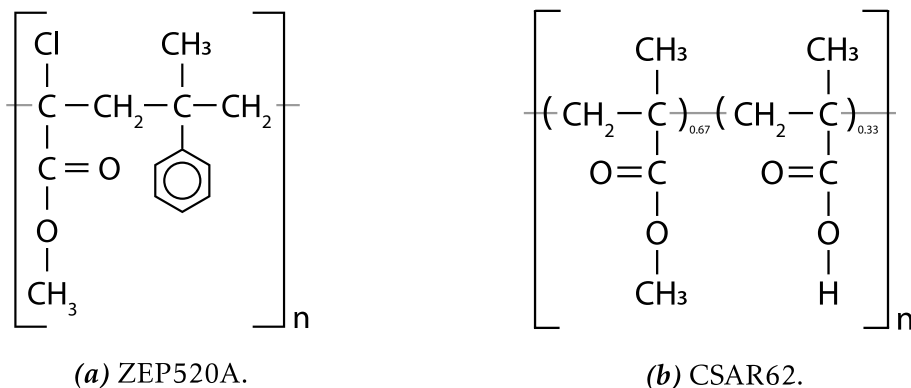
Figure 3.1: Schematic of a PECVD reactor chamber.

The deposition chamber consists of a showerhead, which also serves as a top electrode that is driven by an RF frequency of 13.56MHz, and a grounded bottom electrode which can be heated. Both silicon dioxide (SiO_2) and silicon nitride (SiN_x) can be deposited using the precursor silane (SiH_4) and either ammonia (NH_3) for SiN_x or nitrous oxide (N_2O) for SiO_2. The precursor gasses are introduced into the chamber through the showerhead along with N_2 as a carrier gas, ensuring uniform deposition on the sample. Gas phase by-products are pumped out of the deposition chamber along the bottom electrode.

The reactor used in this thesis is an Oxford Plasmalab System 100. The deposition of SiN_x is done at 300°C at 650mTorr, using a SiH_4:NH_3:N_2 ratio of 16:14:980 sccm with an RF power of 20W. SiO_2 is deposited at 300°C at 1000mTorr, using a SiH_4:N_2:N_2O ratio of 8.5:161.5:710 sccm with an RF power of 20W.

3.1.2 Resist spinning, baking and developing

A resist is a thin layer deposited upon the substrate that is sensitive to light or electrons. Patterns are then written in the resist using lithography, either using a mask when performing photo-lithography or by scanning an electron beam when performing electron beam lithography (EBL). Many different resist compounds exist, and the usage depends on the individual needs of the process. One can generally divide resists in two categories: Positive resists which become soluble when exposed and negative resists that become insoluble when exposed. The two resists used for the process described in sections 3.2.2, 3.2.3 and 3.2.4 (ZEP520A and CSAR62) are positive EBL resists and are based around polymer structures, as shown in Figure 3.2. The high energy electrons break the polymer chain, increasing the solubility in that area.



(a) ZEP520A. *(b)* CSAR62.

Figure 3.2: Structural formula of the positive EBL resists ZEP520A and CSAR62.

Resist is deposited using a spinner, which holds the substrate using a vacuum chuck which rotates to get a thin uniform layer. Depending on the rotational speed the thickness of the resist layer can be tuned. Sometimes a compromise has to be made here; For instance, a thick resist layer will withstand dry etching for a longer time period, but will also decrease the accuracy of the written pattern. After spinning the liquid resist, the substrate is heated to evaporate the solvent, leaving a solid polymer layer.

Once a pattern is written, the resist is developed. This process is simply dissolving the exposed resist in case of a positive resist, or dissolving the non-exposed resist in case of a negative resist. The most common follow up step is etching the written pattern into the layer below (Figure 3.3). Another useful feature is the ability for the resist to be used to perform lift-off. In this application a thin layer of material,

for instance Au, is deposited on top of a developed resist layer. The resist layer is then dissolved, taking with it all the material on top, but leaving Au in the opened pattern (Figure 3.4).

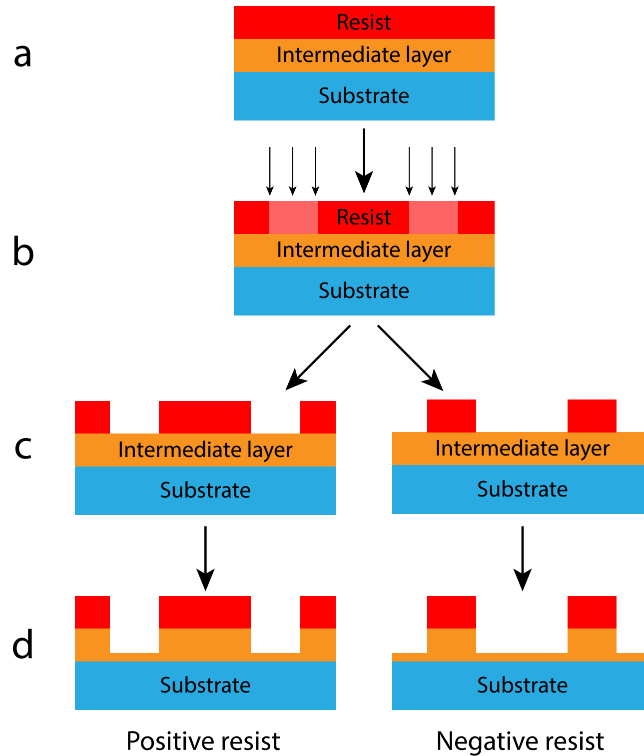


Figure 3.3: (**a**) Substrate prepared for EBL. (**b**) Writing a pattern using EBL. (**c**) Result of development on a positive (left) and negative (right) resist. (**d**) Result of pattern transfer into an intermediate layer.

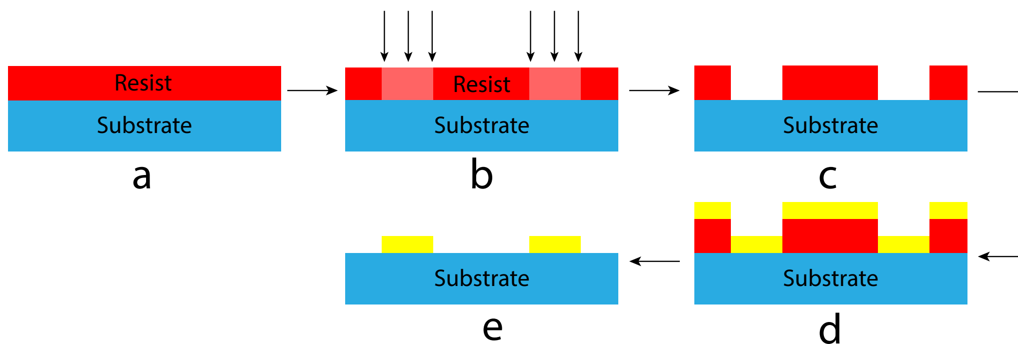


Figure 3.4: (**a**) Substrate prepared for EBL. (**b**) Writing a pattern using EBL. (**c**) Result of development on a positive resist. (**d**) Layer deposition (ex. Au). (**e**) Result of lift-off.

3.1.3 Electron beam lithography (EBL)

Electron beam lithography is one of the most crucial steps during sample processing. It offers a much higher resolution than optical lithography at the cost of very low throughput. It uses a beam of high energy electrons to write patterns into an electron sensitive polymer layer, as explained in section 3.1.2. In Figure 3.5 an EBL system is illustrated. Using electromagnetic lenses the electron beam spot size and beam current is controlled, and beam deflector coils scan the electron beam across the target. The beam blanker is used to deflect the electron beam away from the target during stage movements.

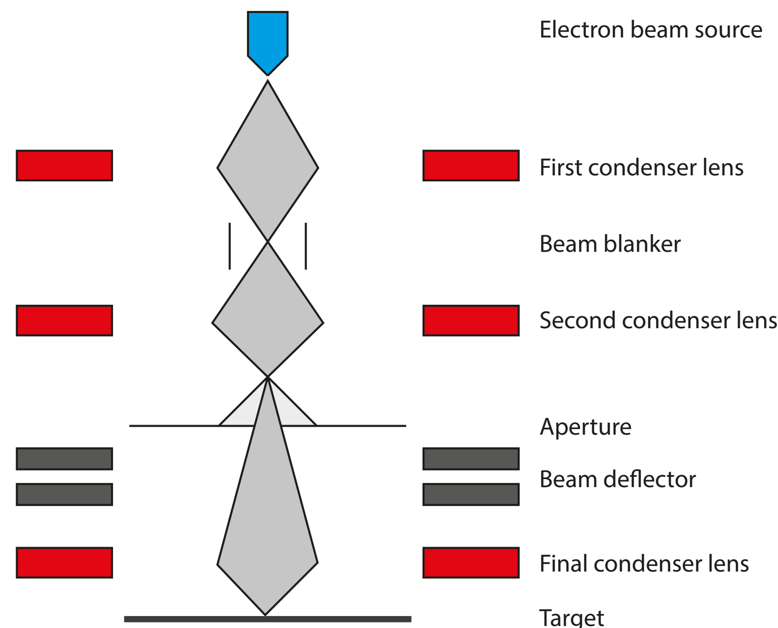


Figure 3.5: Schematic illustration of electron beam lithography. Adapted from [26].

Many effects can be detrimental to the resolution and alignment and need to be considered when using EBL. Drift is important to control when writing a pattern takes a long time. Stage movements can also cause a small misalignment, which can be prevented by fitting a design in an area smaller than the electron beam deflection area.

In the process in this thesis the most important thing is the alignment of subsequent steps. This is done by first etching markers in the substrate which can be found later in the RAITH EBPG 5150 EBL system to align the sample. An example of these markers can be seen in Figure 3.6. The white markers are the main markers which are used in the beginning to properly align and rotate the sample. The secondary markers in yellow surround the actual area where the

pattern will be written. The EBL re-aligns to these markers before the pattern in that area is written to remove any misalignment due to drift. All markers have built in redundancy as each marker is implemented 9 times in a 3 by 3 array. There are two reasons for this; Processing steps in between EBL exposures can damage markers that have already been used, and sometimes some markers don't get properly etched.

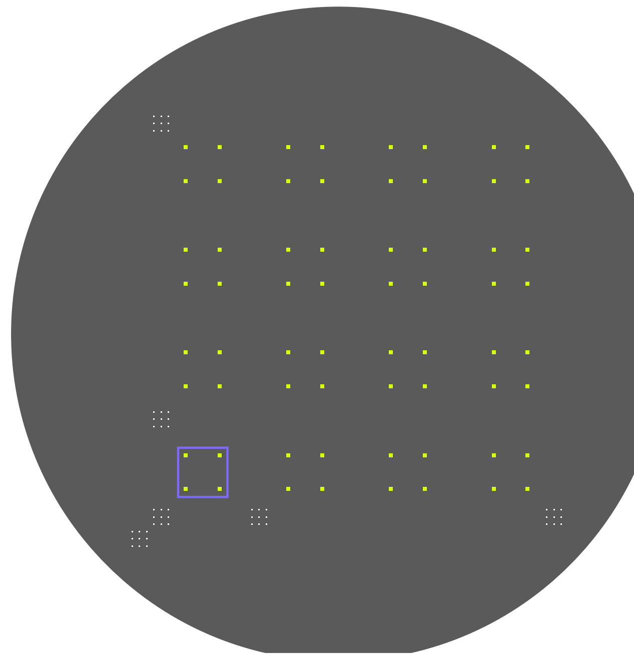


Figure 3.6: EBL pattern with the main markers in white and the secondary markers in yellow.

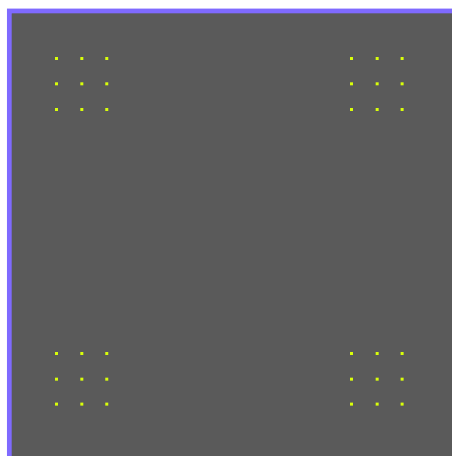
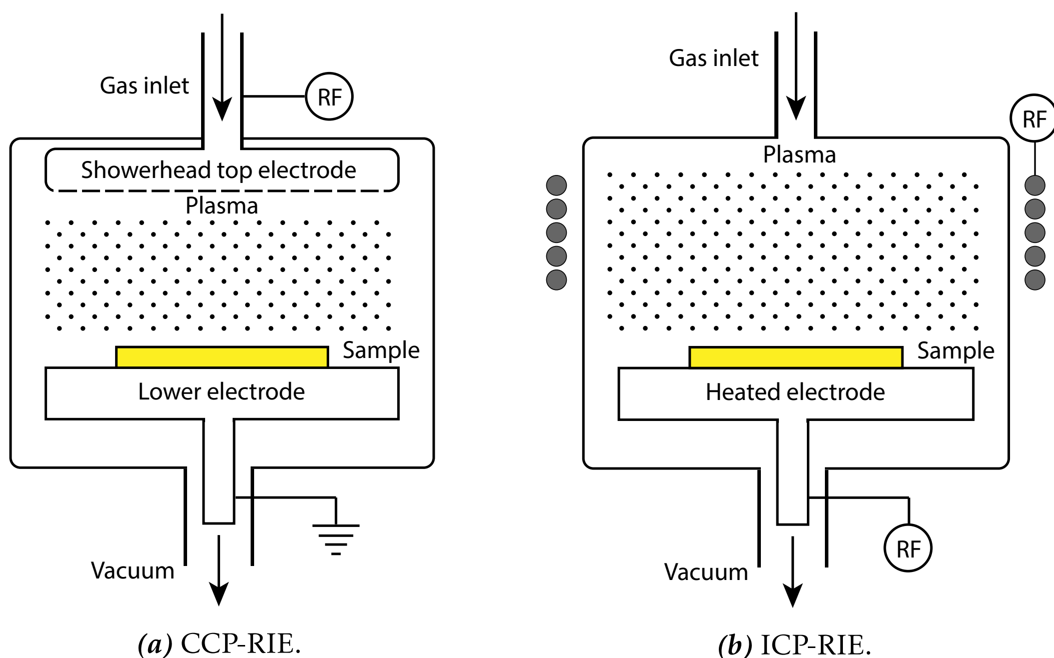


Figure 3.7: Zoom in on the EBL pattern for secondary markers.

3.1.4 Reactive Ion Etching (RIE)

RIE is a technique in which etching can be done anisotropically and at room temperature. A plasma is generated by either a capacitively coupled plasma (CCP) or an inductively coupled plasma (ICP). Reactive species are created in the plasma and are directed towards the sample. Two etching mechanisms play a role; Physical etching and chemical etching. Physical etching is caused by ion bombardment from the accelerated ions, which is an anisotropic process. Chemical etching is caused by desorption of weakly volatile species from the substrate and is thus isotropic [27].

The key difference between a CCP and an ICP is the seperate ICP RF power source connected to the lower electrode, which generates a DC bias that accelerates ions toward the electrode. This means an ICP-RIE can decouple the ion current and energy directed towards the lower electrode.



(a) CCP-RIE.

(b) ICP-RIE.

Figure 3.8

For the etching of SiN_x and SiO_2 an Oxford Plasmalab System 100 CCP-RIE reactor is used, with CHF_3 as etching precursor. Anisotropic etching is desired, so physical etching should dominate. Physical etching is enhanced by a low pressure and high density plasma, which in this case is done using a pressure of 56mT, 100W RF power and a 50:5 CHF_3:O_2 ratio.

3.1.5 Electron beam physical vapor deposition

Electron beam physical vapor deposition is a technique in which high energy electrons are used to heat up and evaporate the target material. This is especially useful for deposition of metals, as the very high melting temperature doesn't allow other kinds of evaporation schemes.

The chamber consists of a cathode with a tungsten filament on which a current is applied, which heats up the filament and emits high energy electrons. An aperture focuses the electron beam, and the beam is directed towards the target using a magnetic field. Once the metal is heated sufficiently it vaporises and coats the substrate in a thin uniform layer. The rate of deposition is measured using a 6MHz crystal of which the frequency decreases when a material is deposited on it. A shutter is kept in front of the substrate until the deposition rate is constant and at the set level, after which the shutter opens and a thin layer can be deposited with high accuracy. The evaporation materials are stored in a water cooled copper plate, which allows multiple metals to be evaporated subsequently without cross contamination during deposition.

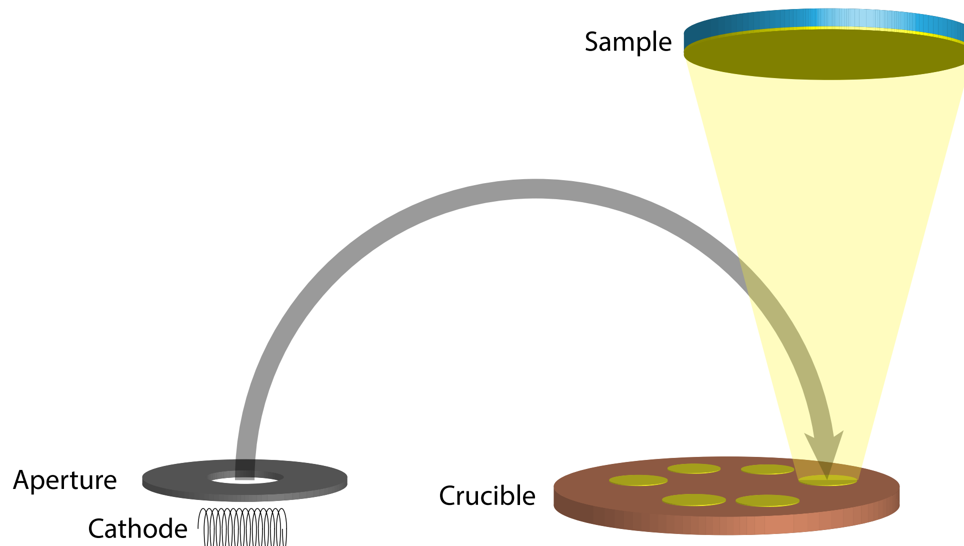


Figure 3.9: Schematic of an electron beam evaporator. The cathode emits electrons which are focused by the aperture and directed towards the crucible by a magnetic field. The metal in the crucible heats up and evaporates, leading to deposition on the sample.

3.1.6 Metalorganic vapor phase epitaxy (MOVPE)

Metalorganic vapor phase epitaxy (MOVPE) is a highly complex growth mechanism capable of producing high quality crystalline layers or structures with abrupt interfaces. It has major advantages compared to molecular beam epitaxy (MBE) in the sense that growth is much faster, more and larger samples can typically be loaded and it does not require ultra-high vacuum, reducing the cost and downtime of the system.

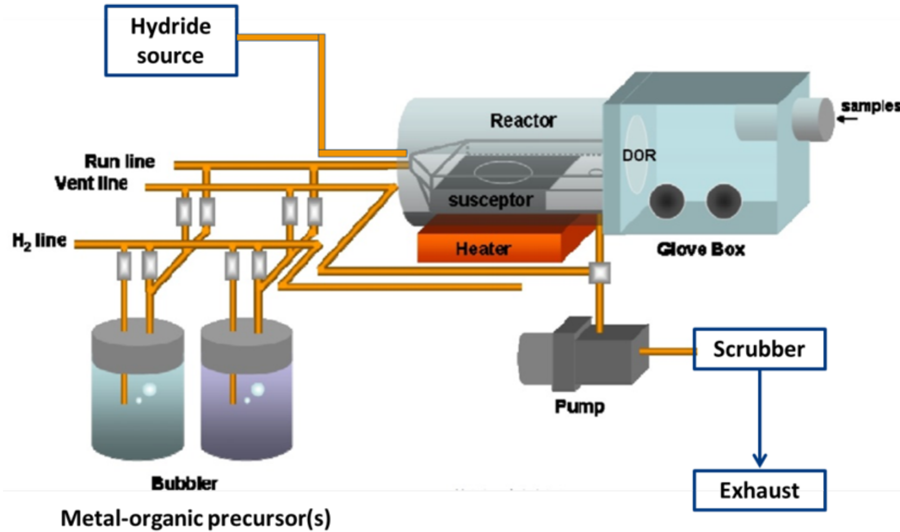


Figure 3.10: Schematic of an MOVPE system [24].

MOVPE is performed in a reactor as show in Figure 3.10. Precursors are introduced into the reactor chamber by a carrier gas, usually hydrogen. In the heated reactor chamber the precursors undergo pyrolysis after which they react at the sample surface. Generally two types of precursors are available; Metal-organic precursors (e.g. tri-methyl In, Sb, Ga) which give the growth scheme its name, and hydride compounds (e.g. PH_3, AsH_3). The hydride compounds are extremely toxic, and are stored in high-pressure metal cylinders which are fed directly to the reactor. The metal-organic precursors are typically volatile and are stored in steel bubblers, either in solid or liquid form. The bubblers have two lines: A supply line that blows the carrier gas through the precursor, creating metal-organic vapor in the process, and an outlet line which transports the metal-organic vapor to the reaction chamber. The amount of precursor introduced to the reaction chamber is determined by the carrier gas flow (f_{tot}), the bubbler pressure (P_{tot}) and the temperature that determines the vapor pressure of the metal-organic source (P_{MO}). The total molecular flow in sccm is given by [28]

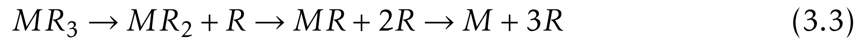
$$f_m = \frac{P_{MO}}{P_{tot} - P_{MO}} \cdot f_{tot} \tag{3.1}$$

Precursor	Melting temp (°C)	a (Torr)	b (Torr·K)	Vapor pressure (Torr) / T (°C)
TMSb	-87.6	7.7068	1697	71.63 / 17
TMIn	88	10.52	3014	0.11 / -10

Table 3.1: Properties of TMSb and TMIn [29]. Vapor pressure is calculated using log(p)=a-b/T and at the temperatures used.

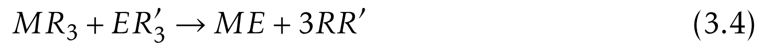
In the reaction chamber, precursors can either fully decompose or decompose stepwise, as given by

$$MR_3 \rightarrow M + 3R \quad (3.2)$$



$$MR_3 \rightarrow MR_2 + R \rightarrow MR + 2R \rightarrow M + 3R \quad (3.3)$$

with M a group III or V element and R either CH_3 or H_2. The general chemical reaction that occurs during MOVPE growth of III/V materials can be written as [29]

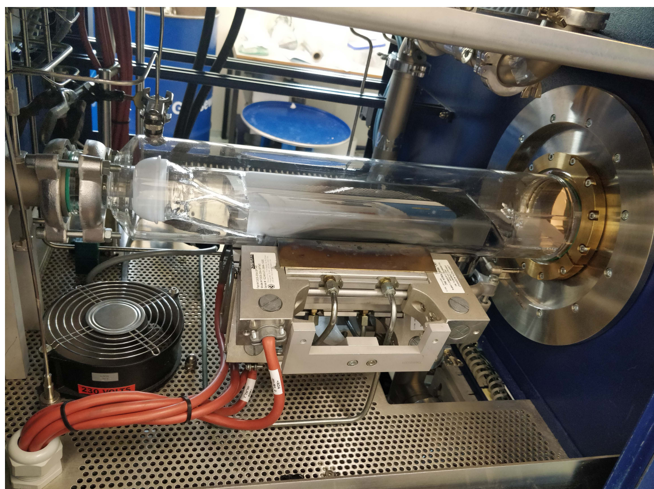


$$MR_3 + ER'_3 \rightarrow ME + 3RR' \quad (3.4)$$

with R and R' either methyl (CH_3) or hydrogen (H_2), M a group III metal and E a group V element. For example, the growth of InP from TMIn and PH_3 precursors can be described by

$$In(CH_3)_3 + PH_3 \rightarrow InP + 3CH_4 \quad (3.5)$$

The reactor chamber is an Aixtron 200 horizontal reactor chamber which consists of a quartz liner which can be seen in Figure 3.11a, and a rotating susceptor which can be seen in Figure 3.11b. The susceptor is suitable for sample sizes up to 2-inch, and has a built in thermocouple to measure the temperature. The susceptor is heated by 5 IR lamps with are located around the bottom half of the quartz.



(a) Aixtron 200 Horizontal reactor chamber.

(b) The susceptor.

Figure 3.11: Images of the Aixtron 200 horizontal reactor. (a) The reactor chamber, which consists of a quartz tube with the bottom half surrounded by 5 infrared (IR) lamps. The gas inlets are located on the left and the outlet to the pump is located on the right. (b) The graphite susceptor with 2-inch sample holder on a rotating disk.

3.1.7 Scanning electron microscopy (SEM)

A scanning electron microscope (SEM) is the most important characterisation tool when growing nanowires. It is used to examine the yield, diameter, length and surface morphology of the nanowires. Instead of photons, like a regular light microscope, it uses a focused beam of accelerated electrons to scan a sample. State-of-the-art SEMs feature a magnification from 10x up to 1000000x, allowing users to image features as small as 0.8 nm [30].

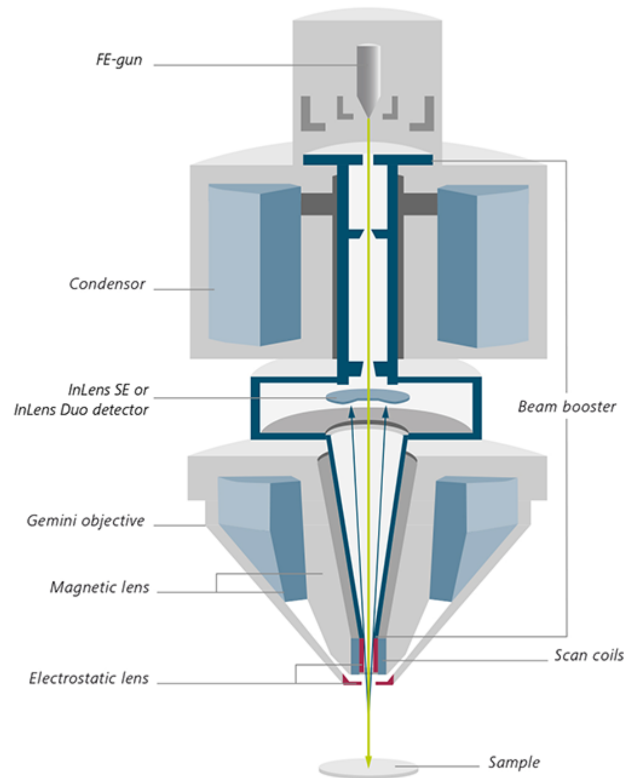


Figure 3.12: Schematic of the Zeiss Gemini objective lens [30].

Primary electrons are generated by a field effect gun which consists of a tungsten filament which is subjected to a very strong electric field, which pulls electrons from the filament. The main method used to create images is by using the secondary electrons, which are electrons ejected during an inelastic scattering event by the primary electron beam from atoms up to 1 µm deep in the sample. Another popular method is by using backscattered electrons. Heavier elements elastically backscatter the primary electrons more than lighter elements, which in turn generates a contrast to form an image.

3.2 Sample preparation

In this section the sample design is discussed as well as the software written for this. Afterwards the sample fabrication process prior to nanowire growth is presented. This process is rather complex and is divided into three distinct phases. Phase one is etching markers in the InP substrate for the alignment of subsequent steps in the EBL. The second phase is etching the trenches that expose the (111)B facets, and the third phase consists of patterning dots with gold catalyst on the trenches.

3.2.1 Design

As the goal of this project is not only to fabricate complex nanowires structures but also to do it in a clever, controllable and reproducible way the first thing to think about is the sample design. The design needs to be flexible, with the option to have many slightly different structures on a sample, for instance different gold droplet diameters, different length of the nanowires or certain distances between merging points. This can be done by hand in software such as AutoCAD, but this would be both tedious and hard to reproduce. So, at the beginning of this project design software was written in the Python programming language, and the basic features will be explained here.

In Figure 3.13 the main user interface of the software is shown and in Figure 3.14 a 3x3 design is shown with the most important parameters indicated. The main design choice is what is called 'Hashtag size', which allows the user to set if they want for instance a nanocross (1x1), a nanohashtag (2x2) or larger structures. The a and b parameters define the distance between the different merging points of the nanowires and the d parameters indicates the depth of the trench, which is directly related to the trench width. The pitch sets the distance between structures, and the dot shift or Δy indicate the offset between the left and the right side of the structure. The c parameter is automatically calculated in the software based on the nanowire length.

Less straight forward parameters are for instance the rectangle width. This parameter determines the width of the trench that gets written in the EBL, which has to be smaller than the actual required width of the trench as the etching process is isotropic. The single shadow and double shadow options can be used to add nanowires slightly in front of the structure, which can cast a shadow on the structure when depositing metal contacts allowing for instance a tunnel barrier as

shown in Figure 2.2.

Behind the scenes a lot of automatic processes happen to make the life of the user easier. As indicated before the structure is automatically optimized using the nanowire length by optimizing the c parameter. The software also checks if the design is larger than the maximum deflection of the EBL system in order to prevent alignment error due to stage movements. It warns the users when the nanowires are too short for the chosen design and when the trenches overlap due to incorrectly chosen a and b parameters.

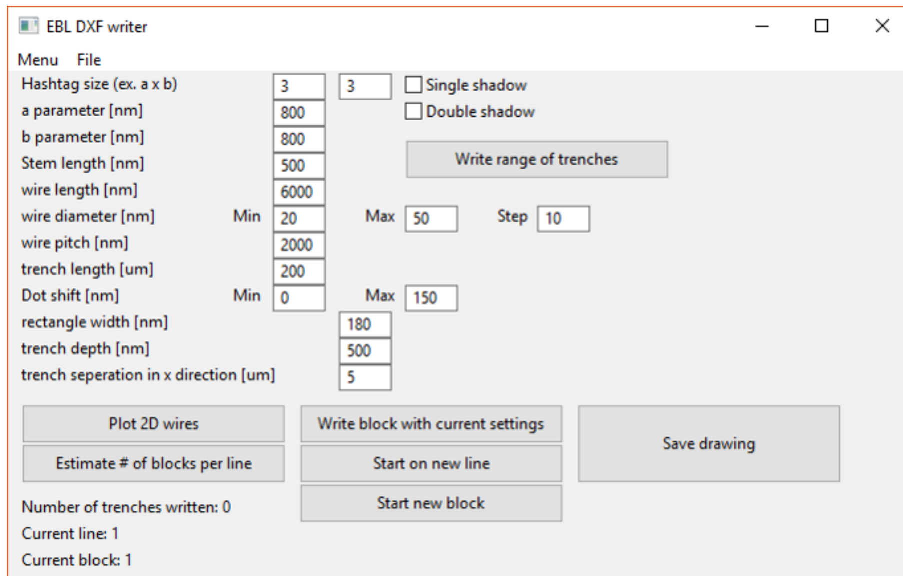


Figure 3.13: The design software main interface.

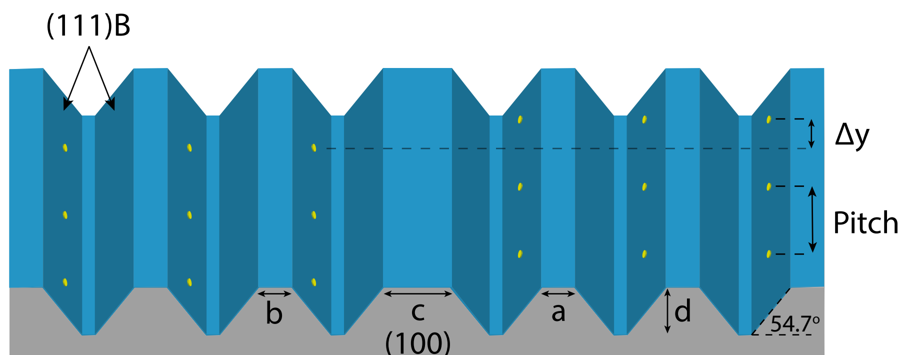


Figure 3.14: Schematic of a 3x3 design. The parameters a and b are equivalent to the parameters in the design software. The trench depth is indicated with d, and c is a parameter automatically calculated in the software. The wire pitch is indicated with pitch and the dot shift is indicated with Δy.

A single sample can have a lot of different structure designs, as indicated in Figure 3.15. Depending on the users needs either identical structures or a range of structures or can be created, with the ability to incrementally change parameters. One thing to keep in mind is that the process requires at least two aligned electron beam lithography (EBL) steps. One for patterning the trenches to expose the (111)B facets and one for patterning the gold droplets on top of the trenches. These steps need to be aligned, which adds another EBL step, patterning markers on the sample for the EBL to align to. Unfortunately the EBL machine is not perfect. Each patterning step can have a small alignment error, according to the machine manufacturer up to 20nm, in practise sometimes even more. To prevent entire samples from being useless due to too large misalignment between the gold droplets and the trenches, a sine function is added to the gold droplets along the trench. This should guarantee that at least a part of the sample has correctly aligned nanowires.

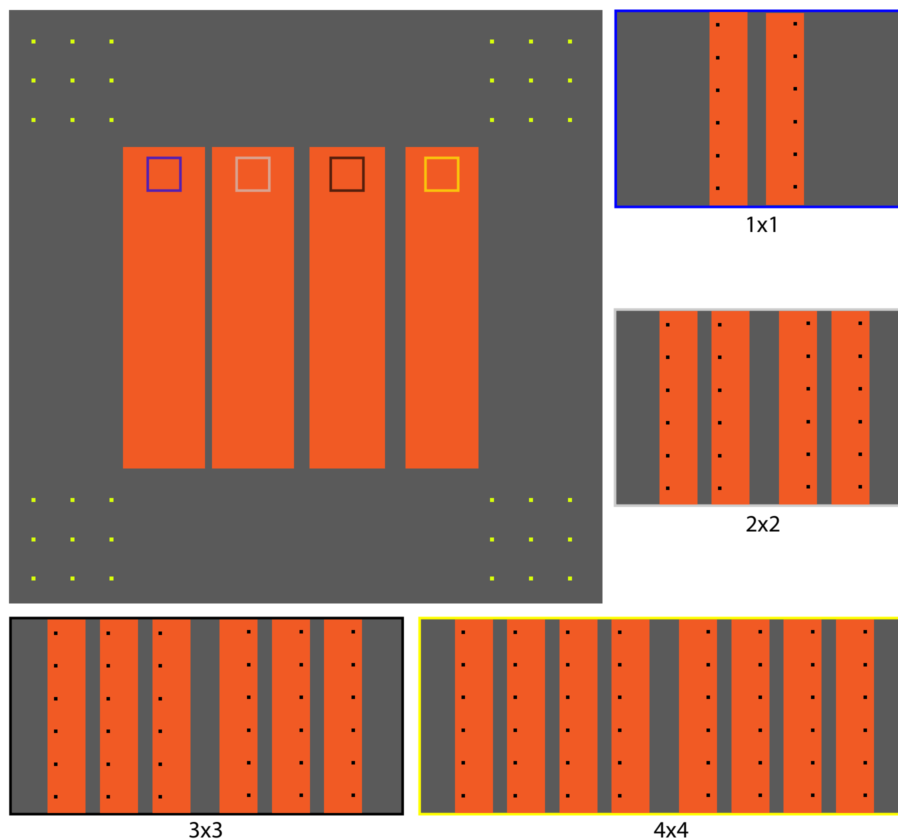


Figure 3.15: Schematic of the EBL pattern used to write the trenches (Orange) and the dots (Black). 4 distinct blocks are present which contain designs for 1x1, 2x2, 3x3 and 4x4 nanowire structures. The yellow squares indicate the local alignment markers.

3.2.2 Phase one: Markers

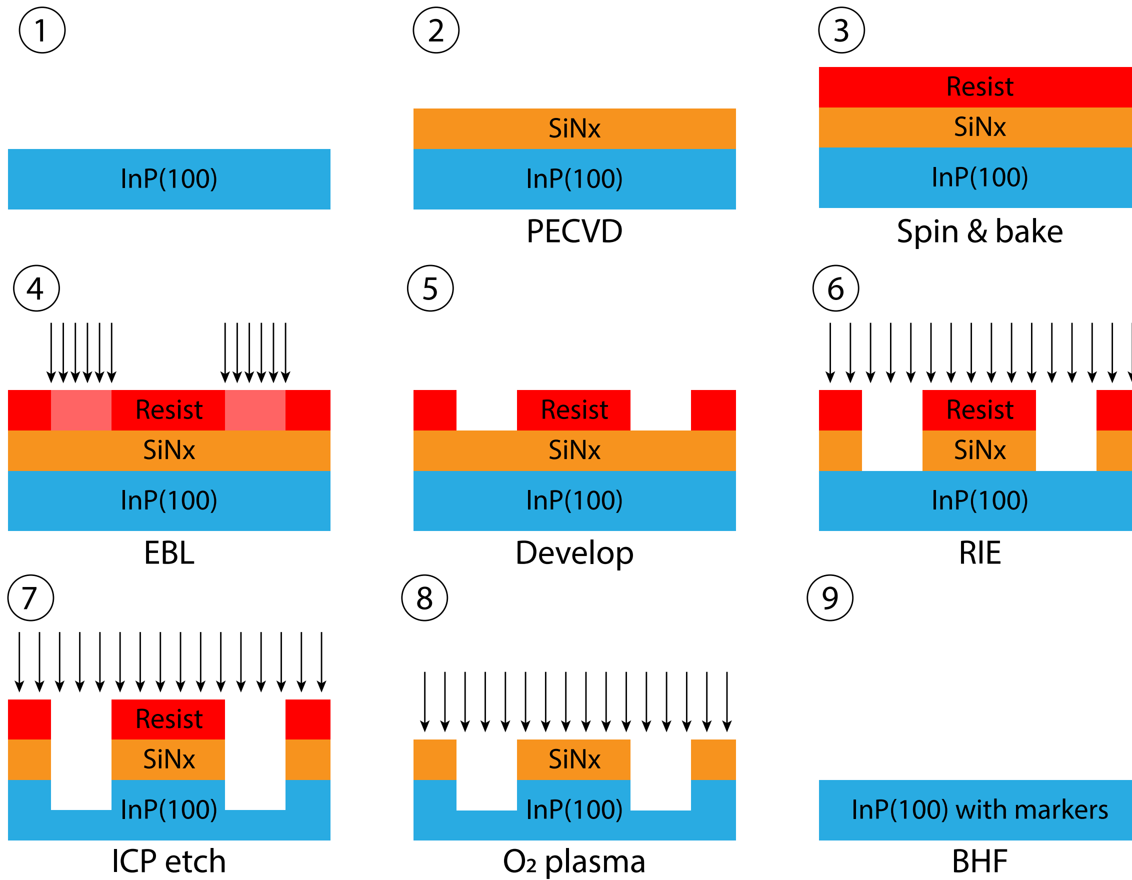


Figure 3.16: Step by step illustration of the process to create alignment markers in a InP(100) wafer. **(1)** A 2 inch InP(100) wafer. **(2)** Using PECVD 100 nm of SiN_x is deposited and the thickness is verified using ellipsometry. **(3)** ZEP520A is spun at 2700RPM for 60s. The resist is then baked on a hot plate starting at a temperature of 100°C which is increased to 200°C in 15 minutes. **(4)** Using EBL markers are patterned. **(5)** The resist is developed using n-amylacetate for 60s, then 45s of MIBK:IPA and 60s of IPA, then blow-dried using N_2. **(6)** Using RIE a 10:1 CHF_3:O_2 plasma is applied for 1:45min to transfer the pattern into the SiN_x layer. **(7)** An O_2 plasma is used to strip away the resist layer. **(8)** Using ICP the pattern is further transferred into the InP wafer. **(9)** The SiN_x layer is stripped by submerging the sample in a 7:1 BHF solution for 3 minutes, then rinsed using UPW and blow-dried using N_2.

Alignment markers such as illustrated in Figure 3.6 are etched into the substrate using the process illustrated in Figure 3.16. The markers are defined in the resist using electron beam lithography (EBL), followed by a reactive ion etch (RIE) which transfers the pattern from the resist layer to the hard mask SiN_x layer. Subsequent ICP etching then transfers the pattern into the substrate.

3.2.3 Phase two: Trenches

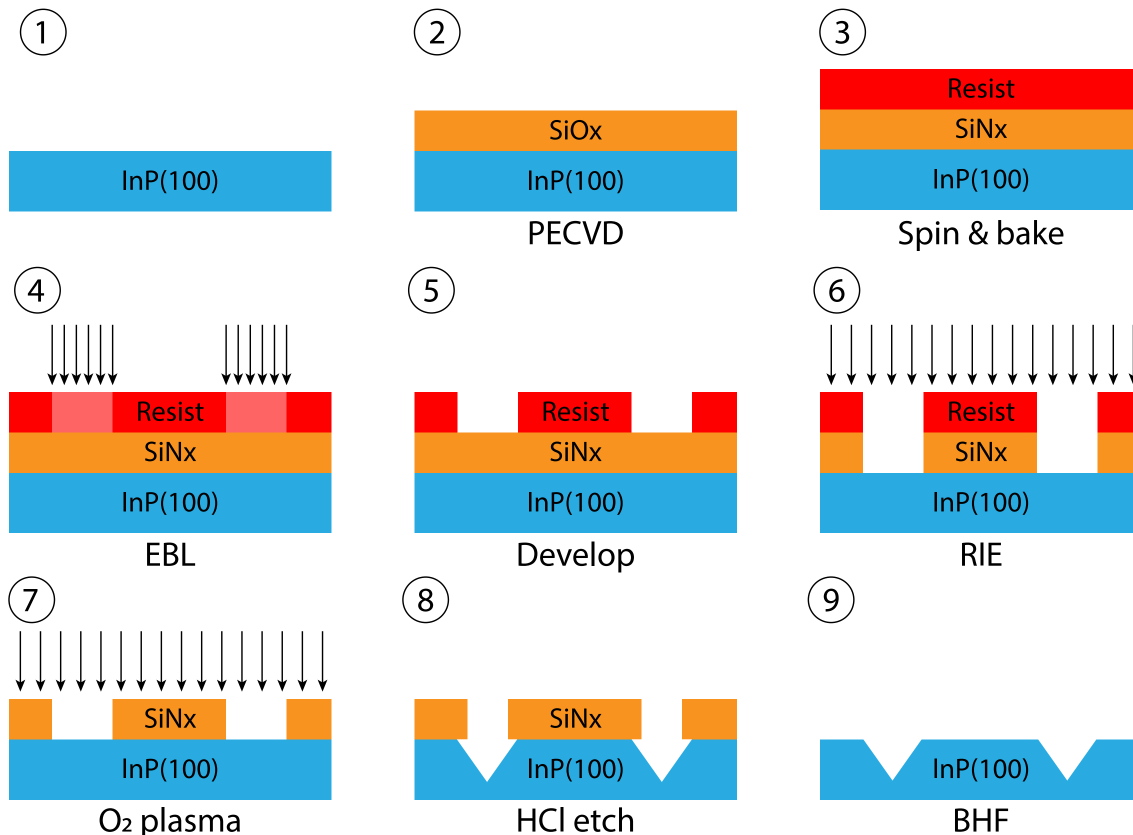


Figure 3.17: Step by step illustration of the process to etch trenches in a InP(100) wafer in order to expose (111)B facets. **(1)** The substrate is submerged in a 10:1 H_3PO_4:H_2O solution for 3 minutes to remove any oxides from the surface, then rinsed using UPW and blow-dried using N_2. **(2)** Using PECVD a 50 nm layer of SiO_x is deposited, and the thickness checked using ellipsometry. **(3)** ZEP520A is spun at 5000RPM for 60s. The resist is then baked on a hot plate starting at a temperature of 100°C which is increased to 200°C in 15 minutes. **(4)** Using EBL the pattern for trenches is written. **(5)** The resist is developed using n-amylacetate for 60s, then 45s of MIBK:IPA and 60s in IPA, then blow-dried using N_2. **(6)** RIE is used to transfer the pattern into the SiO_x layer using a CHF_3:O_2 plasma for 1:15min. **(7)** An O_2 plasma is used to strip away the resist layer. **(8)** The sample is submerged in a 10:1 diluted H_3PO_4 solution for 3 minutes and put in the MOVPE glovebox under N_2 atmosphere to prevent re-oxidation. **(8)** Trenches are etched in the MOVPE using 1% HCl gas. **(9)** The SiO_x layer is stripped using a 7:1 BHF solution for 1 minute, then rinsed using UPW and blow-dried with N_2.

Trench structures that expose the (111)B facets are etched in the substrate using the process illustrated in Figure 3.17. The structures are defined using electron beam lithography (EBL), then transferred into the SiO_x layer using a reactive ion etch (RIE) and etched using an in-situ gaseous HCl etch to expose (111)B facets on

an InP(100) crystal surface.

In Figure 3.18 the recipe for the HCl etching in the MOVPE system is illustrated. The substrate is heated up to 635°C under 1.83E-02 molar flow of PH_3 to prevent P from evaporating from the substrate, which damages the substrate and allows In droplets to form. A molar flow of 3.33E-04 of vapor phase HCl is then introduced into the reactor which selectively etches the exposed (100) substrate, leaving (111)B faced trenches in the substrate. The substrate is then cooled down under PH_3 flow to prevent any evaporation.

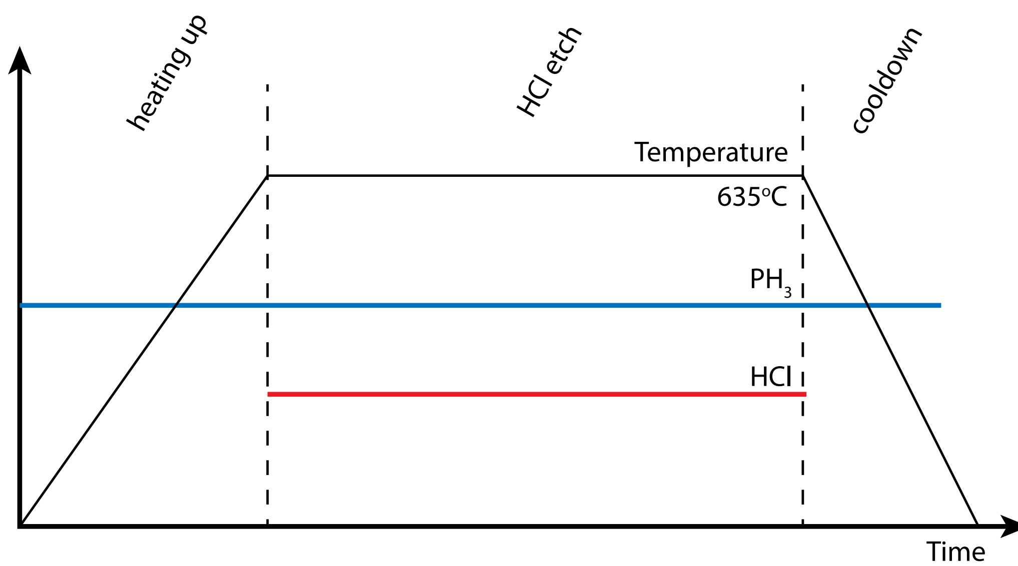


Figure 3.18: Schematic of the etching of V grooves in InP(100) using HCl.

3.2.4 Phase three: Dots

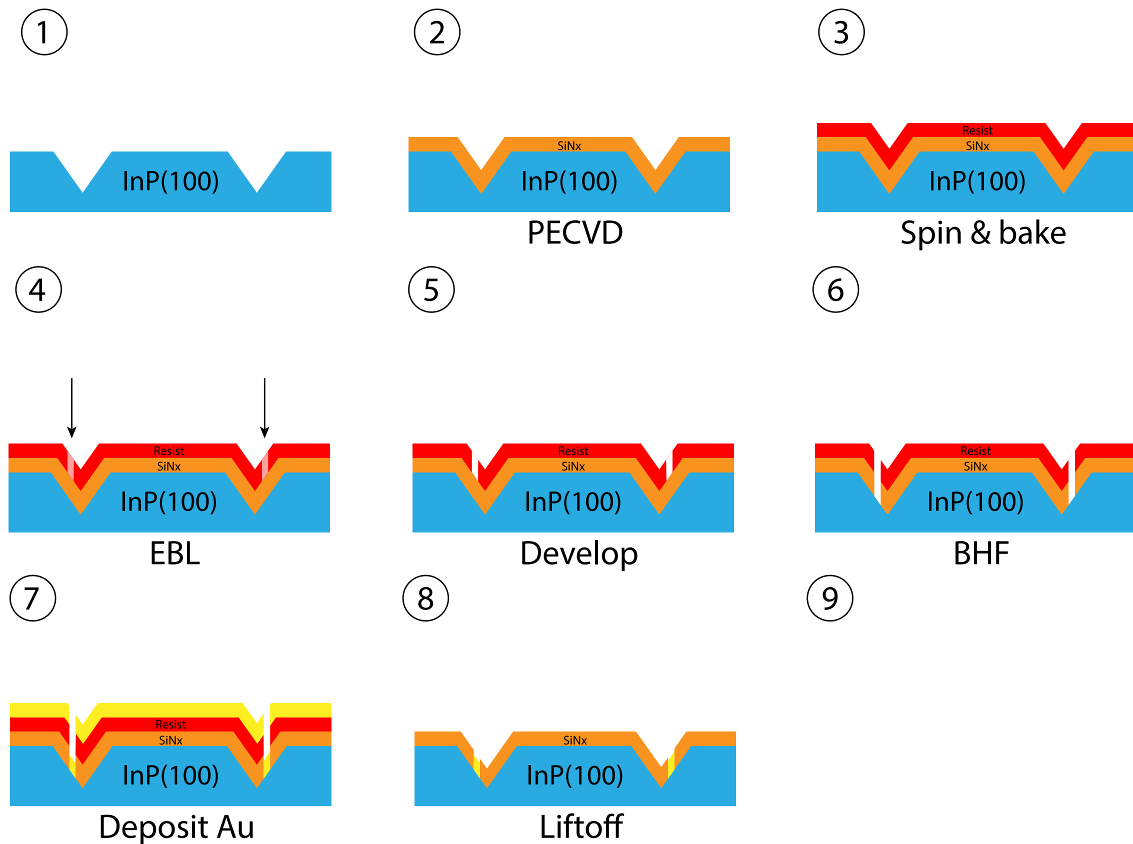


Figure 3.19: Step by step illustration of the process to deposit Au droplets inside specific locations in a SiN_x mask. **(1)** The sample is submerged in a 10:1 diluted H_3PO_4 solution to remove any oxides from the surface, then rinsed with UPW and blow-dried with N_2. **(2)** Using PECVD a 20 nm layer of SiN_x is deposited. **(3)** EBL resist CSAR62 is applied, by first spinning CSAR primer at 4000RPM for 60s and baking it at 180°C for 2 minutes. CSAR62 resist is spun at 4000RPM for 60s and baked at 150°C for 3 minutes. **(4)** With EBL the dot pattern is written. **(5)** The resist is developed using CSAR developer for 60s, CSAR stopper for 30s followed by 60s of IPA and blow-drying with N_2. **(6)** Using 7:1 BHF for 14s the pattern is transferred to the SiN_x. **(7)** 8 nm of Au is deposited. **(8)** Lift-off is performed by submerging the sample in PRS-3000 for at least 30 minutes, then put in acetone and vibrated using ultrasound for 10 minutes. The sample is rinsed using IPA and blow-dried with N_2.

Gold particles with a diameter of 40-50nm, crucial for an optimized nanowire merging process, are positioned on the exposed (111)B facets using the process illustrated in Figure 3.19. The gold droplets act to catalyse nanowire growth via the vapour-liquid-solid (VLS) mechanism as illustrated in Figure 2.11.

Chapter 4

Results

In this chapter the results of stemless InSb nanowire growth as well as InSb nanowire on InP stem growth is presented. First the developed MOVPE growth recipes are explained as a background for all the performed experiments. Then the experiments done in order to grow InSb nanowires without using InP stems at all are discussed. Then the growth of InSb nanowires on InP stems is presented. First the influence of the InP stem length is investigated, after which encapsulation of the InP stem with InSb is optimized. Then a three step InSb growth recipe is developed, which is used to increase the InSb nanowire length while suppressing radial growth. Finally the merging behaviour of the nanowires is discussed.

4.1 Growth recipes

4.1.1 InP nanowires

In Figure 4.1 the general growth recipe for InP nanowires is shown. First the sample is heated up to 510°C and annealed for 5 minutes to remove any remaining oxides. This is done under PH_3 flow, to prevent P evaporation and subsequently the formation of In droplets on the substrate. The temperature is then reduced to the growth temperature of 450°C and the PH_3 flow is stopped for the filling step, which allows the gold catalyst droplet to be saturated with In. Then the PH_3 is opened again after which epitaxial InP nanowire growth starts. Once the growth is done the In flow is stopped, and the sample is cooled down under a PH_3 flow to prevent evaporation. Different lengths of InP nanowires are grown, which will be discussed in section 4.3.

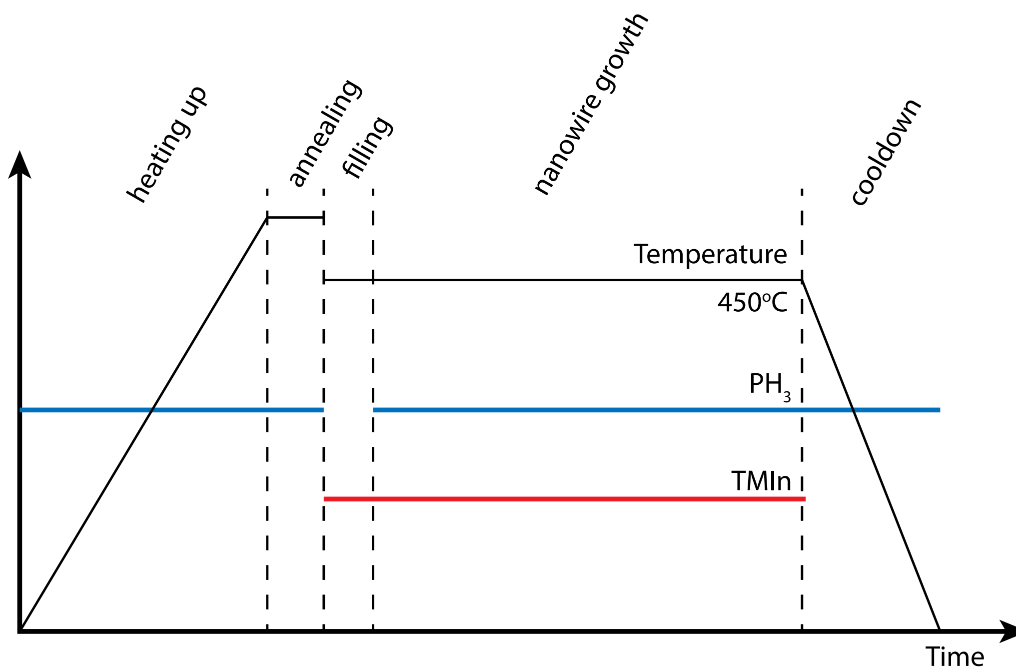


Figure 4.1: Schematic of the growth of InP nanowires.

4.1.2 InSb nanowires

In Figure 4.2 the general growth recipe for InSb nanowires is shown. First the sample is heated to the growth temperature. The heating is done under AsH_3 flow to prevent P evaporating from the substrate from entering the catalyst droplet. The AsH_3 is then flushed away. The gold catalyst droplet is then filled with In, after which the Sb flow is opened and nanowire growth starts. For stemless InSb a single growth step is used, which is also used in the beginning for InSb on InP stems. Later a two-step process is used, and finally a three-step growth process is developed which is explained in section 4.3. After growth the In flow is stopped and the sample is cooled down under Sb flow to prevent any evaporation.

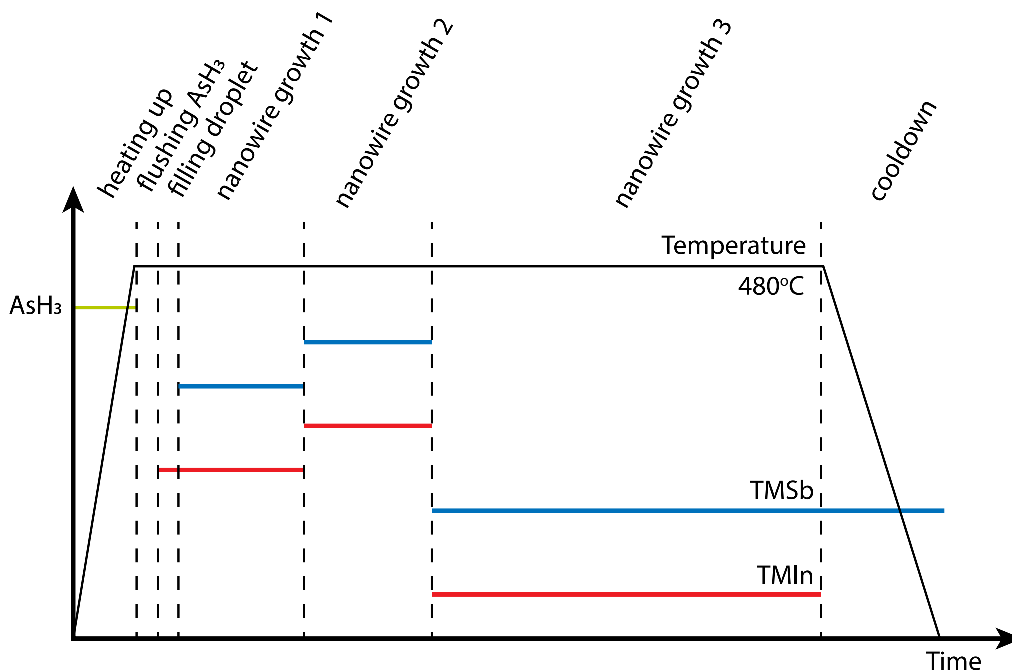
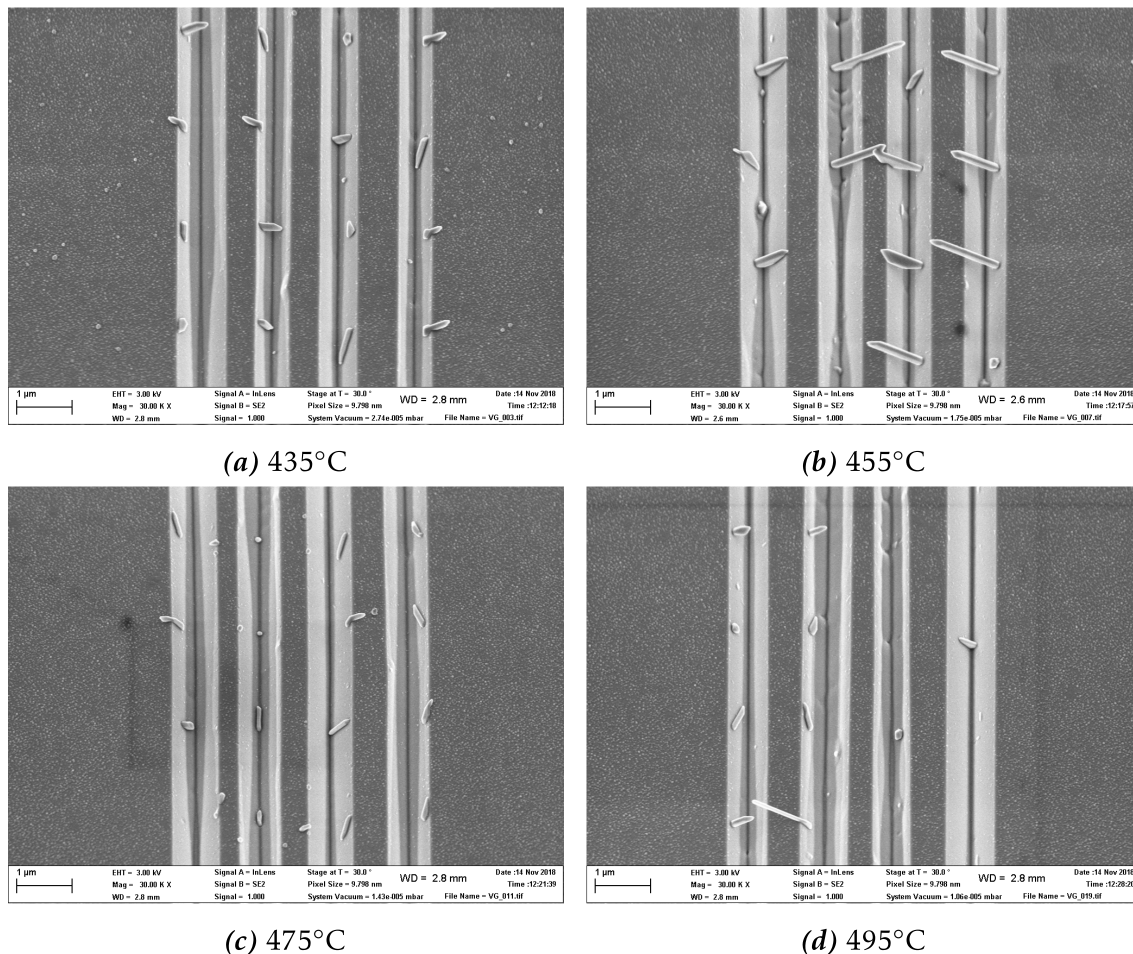


Figure 4.2: Schematic of the growth of InSb nanowires. For stemless InSb nanowires one growth step is used, while for InSb on InP stems a single step, a two-step and a three-step process is used.

4.2 Stemless InSb nanowires

The first thing to investigate for nanowire growth is the growth temperature. The temperature must be sufficiently high to heat the gold catalyst particle above its eutectic point of 337°C to allow a liquefied Au-InSb alloy [31], as well as to efficiently crack the precursors. However, too high a temperature can damage the sample or change the growth regime. To find the optimum temperature a series is performed, with a constant TMIn and TMSb flows for one hour and a temperature ranging from 435°C up to 495°C. In Figure 4.3 SEM images of the resulting growth are shown. At 435°C, 475°C and 495°C the yield and length of the nanowires is very low, with almost all nanowires growing in-plane in the trench. At 455°C the yield is slightly improved, with occasional nanowires growing in the intended direction.



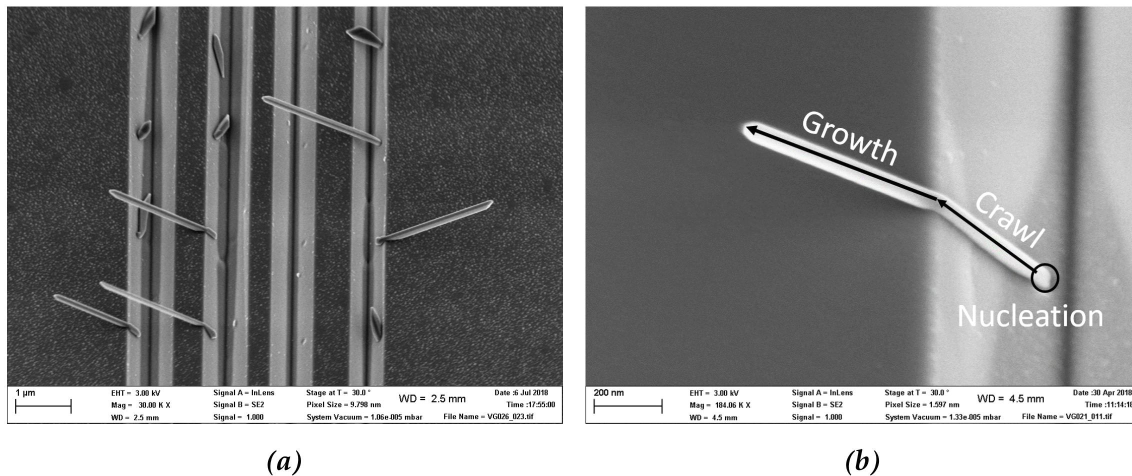
(a) 435°C *(b)* 455°C

(c) 475°C *(d)* 495°C

Figure 4.3: SEM images of 4 growth recipes with the only difference the growth temperature. A molar fraction of 1.95E-07 of TMIn and 1.23E-04 of TMSb is used.

It is observed that 455°C gives the highest yield, so the rest of the experiments on stemless InSb are preformed at this temperature. In table A.1 and A.2 a list of all the stemless InSb growth experiments is shown.

Despite the large range of TMIn and TMSb flows as well as wildly different V/III ratios no conditions are found for proper nanowire growth. However, two effects are consistently observed, which can be seen in Figure 4.4 and 4.5. In Figure 4.4 the nanowires are seen growing in-plane on the inclined facets. Once the (100) crystal plane is reached the growth continues bottom-up, but in the other (111)B direction. This behaviour is seen on almost all samples, being more common on samples where lower flows are used. This 'crawling' behaviour might be caused by the InP substrate, which can supply P to the gold catalyst droplet. The P is known to prevent Si absorption in the catalyst particle [32], which might also be the case for Sb. This can allow the In filled Au droplet to 'walk' along the trench. In theory this effect can be used to form nanowire structures by placing the gold droplets on the opposite side of the trench. However, the merging quality of these nanowires is not clear yet, which will be investigated in the future using cross-sectional TEM.

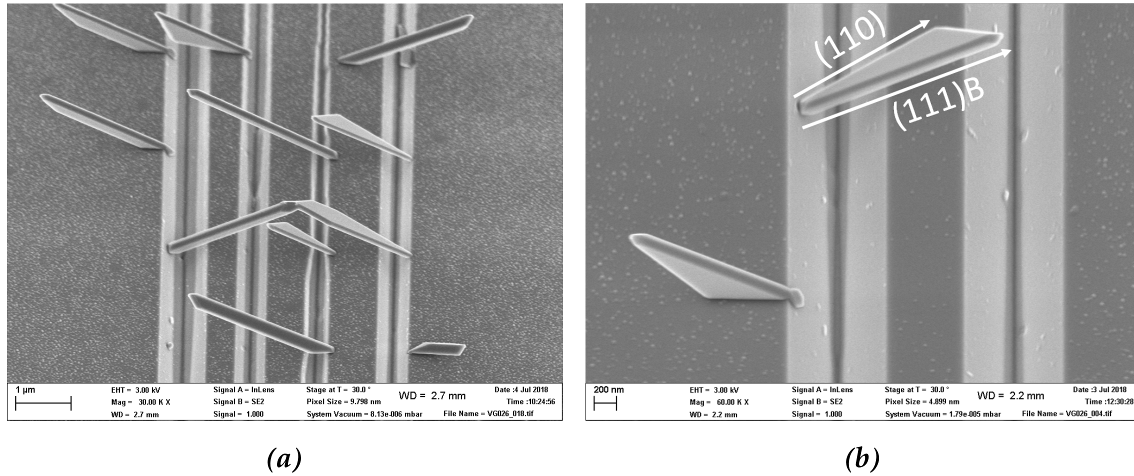


(a) *(b)*

Figure 4.4: SEM images after stemless InSb growth showing nanowires growing in-plane on the inclined (111)B facets. Once the (100) crystal plane is reached the nanowires continue growing bottom up in the other (111)B direction.

In Figure 4.5 flake forming can be seen. These kind of flakes are seen on all samples with high flow, getting progressively larger and more abundant. Due to the high supersaturation induced by the high flow, the gold droplet is enlarged which allows the formation of high energy nucleation points, on which catalyst free growth can occur. This catalyst free growth happens in the (110) direction, and combined with the VLS growth in the (111)B direction a flake forms [33]. An

interesting observation is that some of the nanowires/flakes grow in the intended (111)B direction. A possible explanation is that the high flow forces quick epitaxial growth which prevents the gold catalyst from crawling along the trench.



(a) *(b)*

Figure 4.5: SEM images showing flake growth due to high precursor flows. VLS growth occurs in the (111)B direction and VS growth in the (110) direction.

4.3 InSb nanowires on InP stems

4.3.1 InP stem length and encapsulation

As shown in the previous section, the growth of stemless InSb nanowire networks is challenging. This is because of the crawling effect of the nanowires, in which the nanowires grow in-plane on the inclined facets. To overcome this challenge the following experiment is designed. The idea is to use as short as possible InP stems, on which InSb is grown epitaxially. The aim is then to encapsulate the InP stems with InSb to prevent the InP from evaporating. It is important to do this while getting a high yield as well as a large aspect ratio in order to get suitable nanowire networks. The intended encapsulation process is illustrated in Figure 4.6.

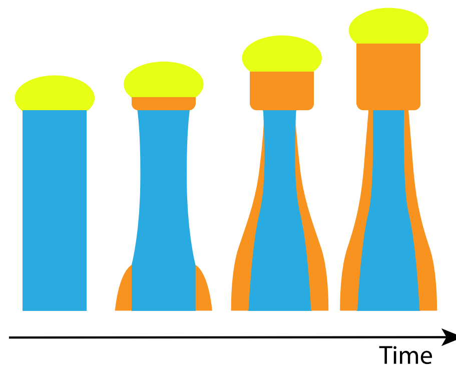
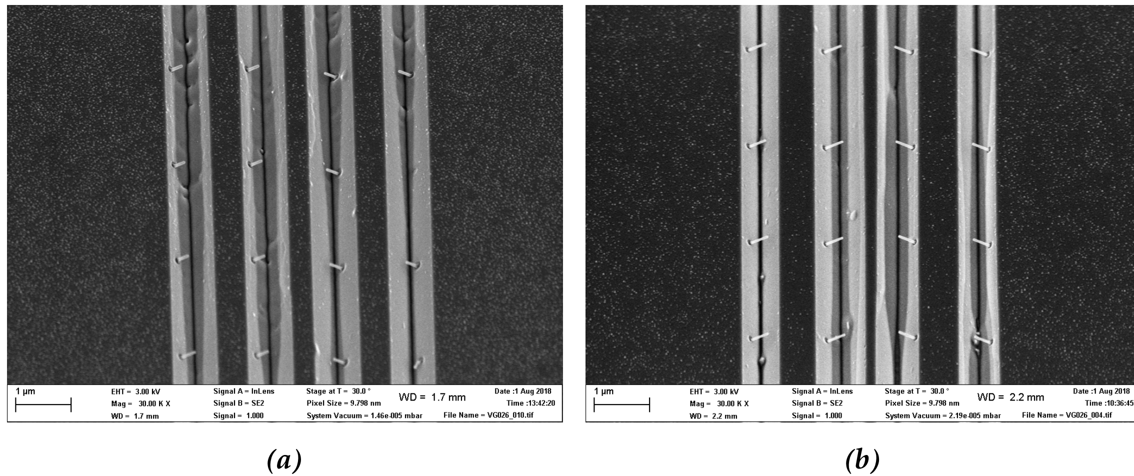


Figure 4.6: Schematic of the intended encapsulation of the InP stem. The blue represents the InP stem, orange represents the InSb. Yellow represents the gold droplet.

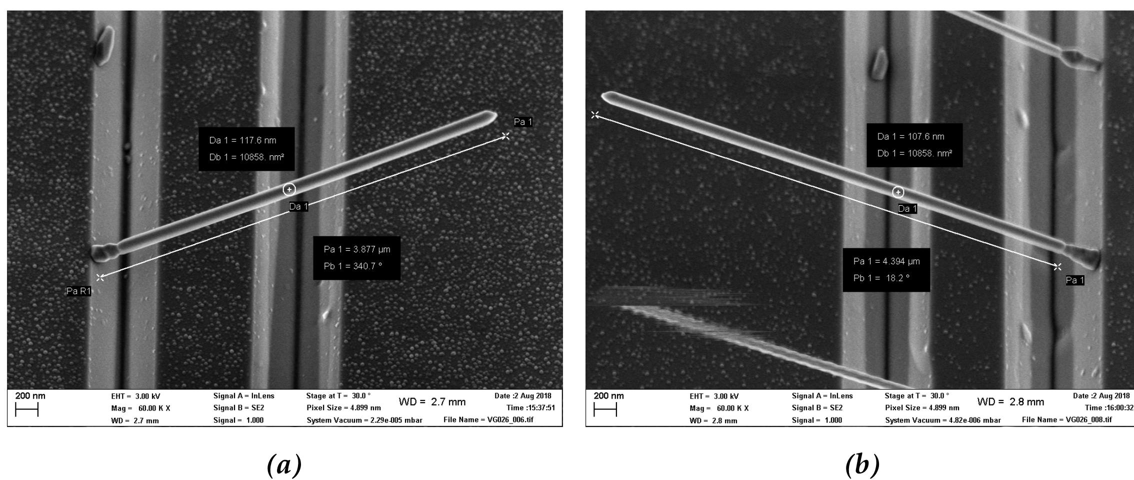
The first thing to investigate is the effect of the InP stem length on the InSb nanowire growth. In Figure 4.7 InP stems can be seen which are grown for (a) 2 minutes and (b) 3 minutes. The stems have a high (estimated >80%) yield, with the stem length directly proportional to the growth time with (a) 200nm and (b) 300nm length.



(a) *(b)*

Figure 4.7: SEM images of (a) InP stems grown for 2 minutes resulting in 200nm long stems. (b) InP stems grown for 3 minutes resulting in 300nm long stems.

In Figure 4.8 InSb is grown on top of the InP stems. InSb growth is done for 1 hour at 495°C on top of (a) 200nm InP stems and (b) 300nm InP stems.



(a) *(b)*

Figure 4.8: SEM images of (a) InSb grown on top of 200nm long stems and (b) InSb grown on top of 300nm long stems. Growth is done with molar fractions of 4.18E-07 of TMIn and 3.10E-04 of TMSb and a temperature of 495°C.

The stems are fully encapsulated for both the stem lengths, and the nanowires in (b) are overall slightly longer and thinner than in (a). The yield of properly grown nanowires on both samples is low (estimated <10%), which is caused by most of the InP stems getting encapsulated or overgrown too quickly which prevents proper InSb nanowire growth. This is illustrated in Figure 4.9 with properly grown and encapsulated nanowires on the right and overgrown stems on the left.

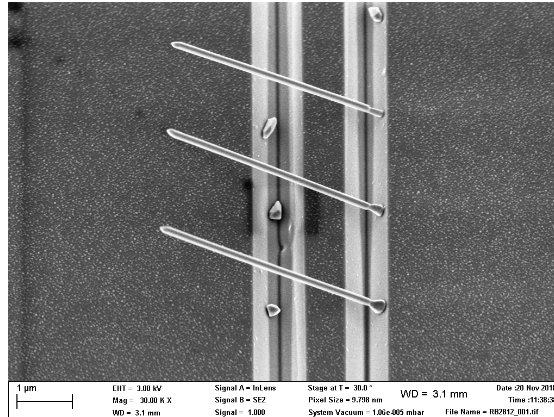
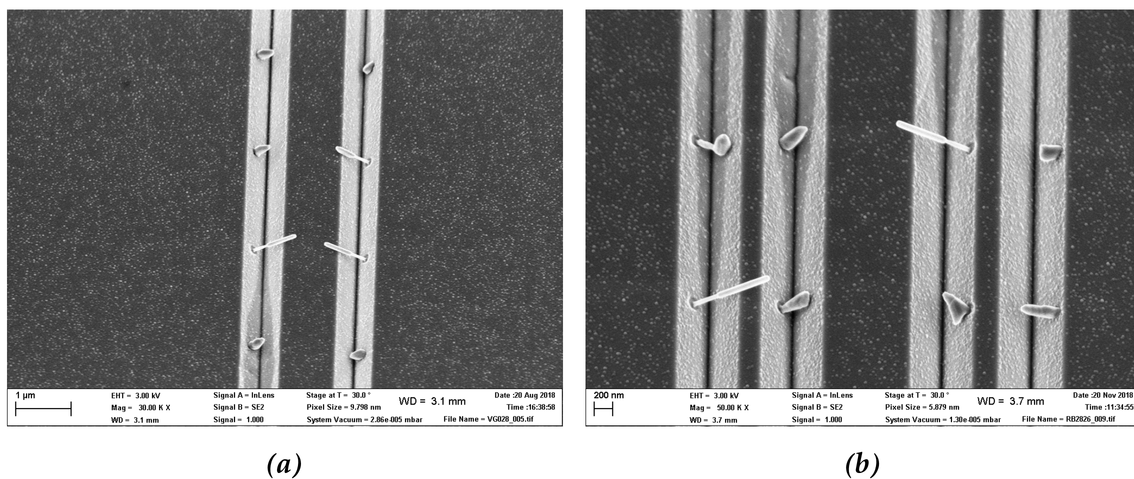


Figure 4.9: SEM image with properly grown and encapsulated nanowires on the right and overgrown stems on the left.

To investigate the InP stem encapsulation in more detail the same growth recipe is repeated with 200nm long InP stems, now with a short growth time of 15 minutes. In Figure 4.10 the resulting nanowires can be seen. It is observed that most of the InP stems already get overgrown in the first 15 minutes of growth, which prevents proper InSb growth. The nanowires that do grow have no encapsulation on the stem at all, which indicates that the InSb did not have enough time to nucleate on the stem.

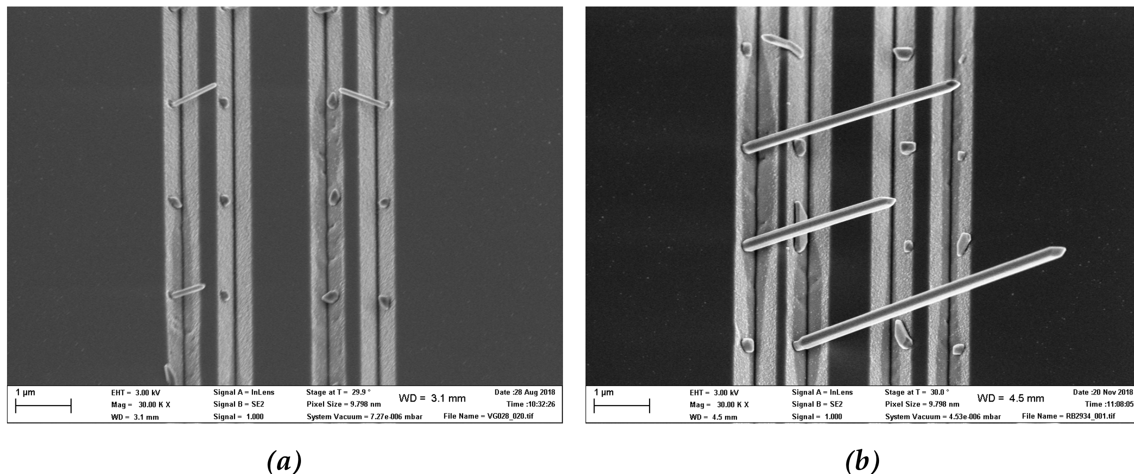


(a) *(b)*

Figure 4.10: SEM images of InSb grown on 200nm InP stems for 15 minutes. Growth is done with molar fractions of 4.18E-07 TMIn and 3.10E-04 TMSb and a temperature of 495°C.

Two possible solutions come to mind to prevent InP stem overgrowth that occurs in the first 15 minutes of growth. Either the encapsulation rate needs to be decreased to allow the InSb nanowire to grow longer before they are encapsulated, or longer stems need to be used.

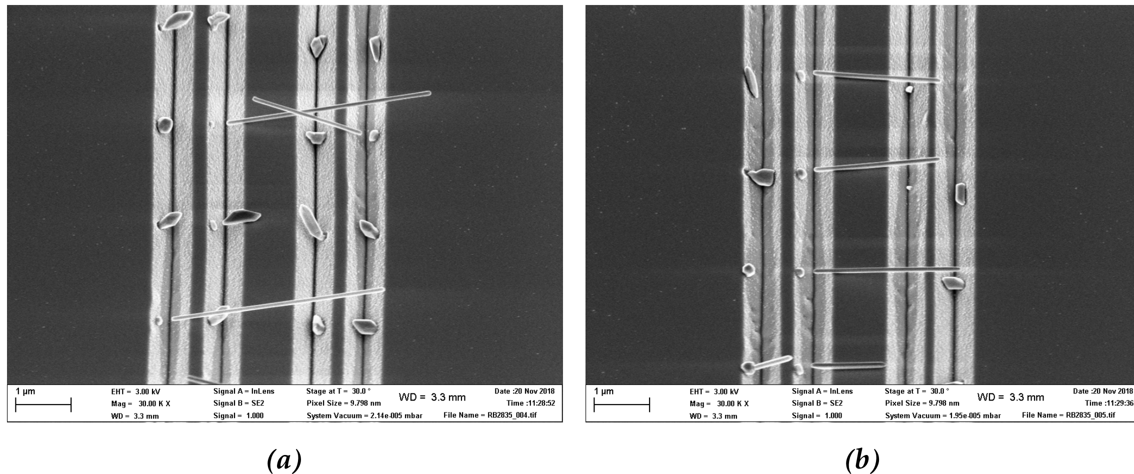
First an attempt is made to decrease the encapsulation rate. In Figure 4.11 the TMIn and TMSb molar fractions are lowered by a factor of 2 in comparison to the previous experiment, and again the InSb growth is done on InP stems of 200nm. This gives a similar result as before, with most of the InP stems getting overgrown. The nanowires that do grow properly have fully encapsulated stems. A possible explanation is that the nanowires that grow properly have relatively late InSb nucleation on the stem, allowing the InSb nanowire to grow before encapsulation.



(a) *(b)*

Figure 4.11: SEM images of InSb grown on 200nm InP stems for (a) 1 hour and (b) 3 hours. Growth is done with molar fractions of 2.09E-07 TMIn and 1.57E-04 TMSb and a temperature of 495°C.

In Figure 4.12 the molar fraction is lowered by a factor 2 again. It is observed that still a lot of stems are overgrown, and that the nanowires with late nucleation on the stem now show full evaporation of the stem. This causes either nanowires standing up without a stem as in Figure 4.12a or fallen over nanowires as in Figure 4.12b.



(a) *(b)*

Figure 4.12: SEM images of InSb grown on 200nm InP stems for 3 hours. Growth is done with molar fractions of 1.12E-07 TMIn and 7.85E-05 TMSb and a temperature of 495°C.

Lowering the flow did not have the intended effect. The stems are too short and have an unreliable nucleation rate of InSb on the InP stem. This leads mostly to overgrown stems, even at very low precursor flow rates. The nanowires that do grow have either no or late InSb nucleation on the stem, which prevents overgrowth but allows the stem to fully evaporate. An explanation for the high overgrowth of the stems even at low flow is that diffusion on the substrate increases at low flow. This leads to an increased effective flow at the InP stem, which in turn can promote the overgrowth.

The next step then is to investigate the use of longer InP stems in order to prevent the overgrowth. In Figure 4.14 InP stems are grown with a length of 500nm. The InP yield is high (estimated >90%), and the length is similar all over the sample and for the different structures sizes, as can be seen in Figure 4.13. This length should allow the InSb nanowire enough time to grow before the InP stem is fully encapsulated.

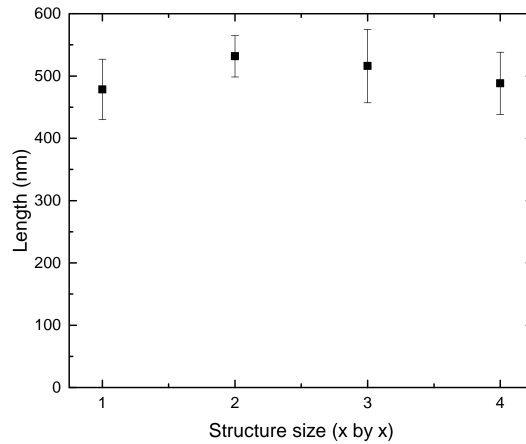
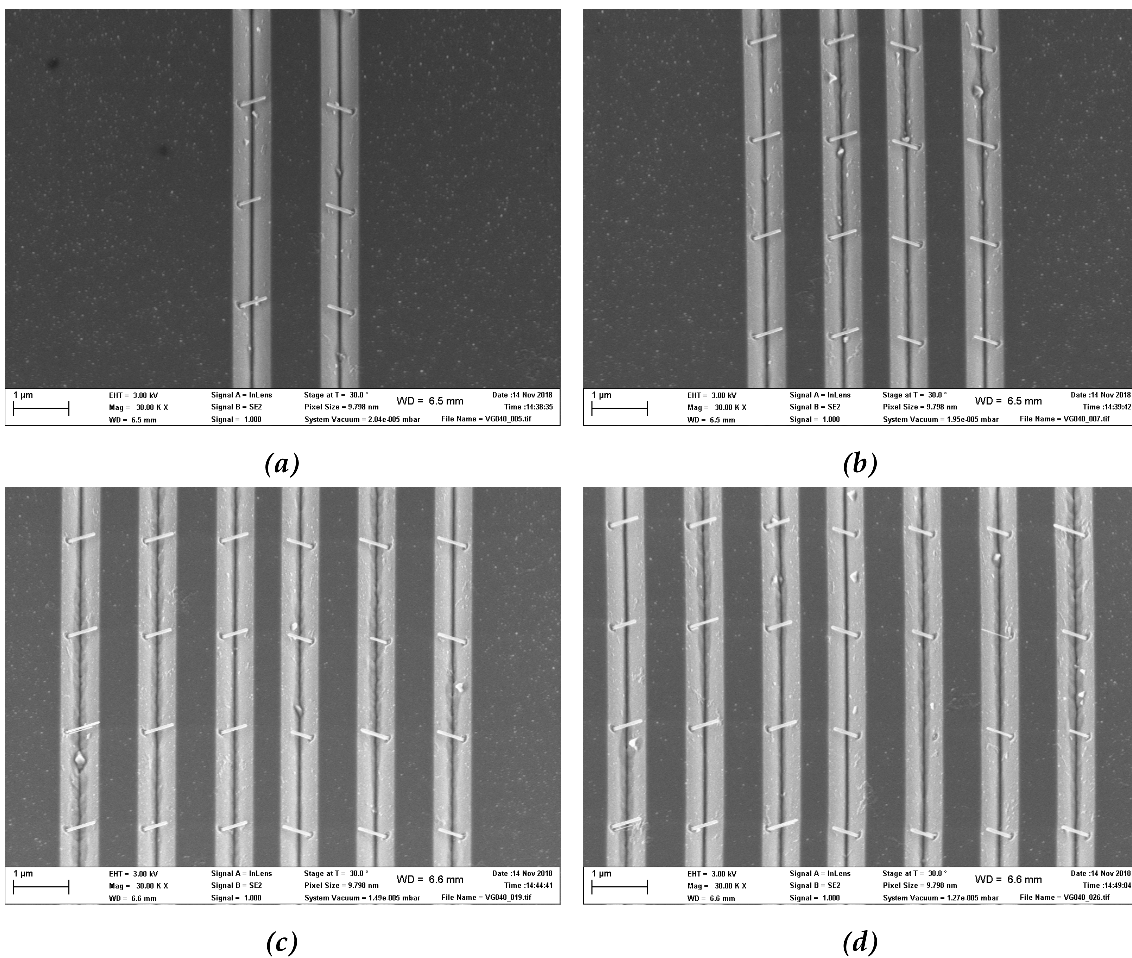


Figure 4.13: Graph showing the InP stem length for different structure sizes.



(a) ***(b)*** ***(c)*** ***(d)***

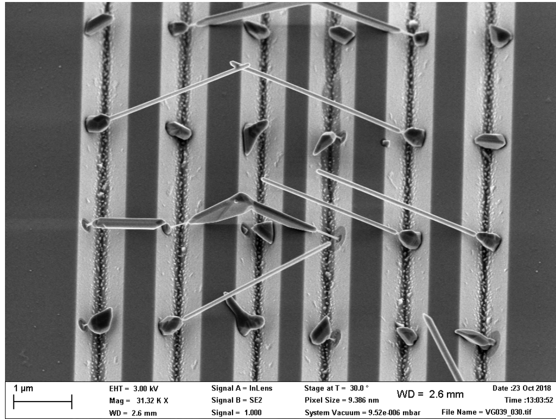
Figure 4.14: SEM images of 500nm long InP stems for (a) 1x1 nanocrosses, (b) 2x2 hashtags, (c) 3x3 and (d) 4x4 structures.

To have more control over the encapsulation of the 500nm long stems a two-step growth process of InSb is introduced. First a medium flow is used in order to grow the InSb nanowires in both length and to start encapsulation. The molar fraction is then increased to a high flow in order to promote even more encapsulation. At the same time the growth temperature is decreased slightly from 495°C to 480°C to decrease the InP evaporation rate. It is known that there is a possible difference in precursor decomposition when the temperature is lowered, so these experiments cannot be directly compared to the growth experiments discussed before. In table 4.1 a list of growth experiments is shown. The total InSb growth time is 40 minutes, with varying times for the medium and high flow steps in order to find the right balance. The resulting nanowire length and diameter are indicated.

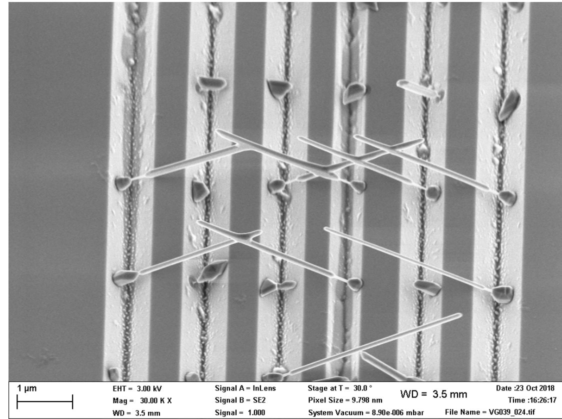
Run	Medium flow time	High flow time	Diameter	Length
RB2929	20min	20min	80 - 100 nm	3 - 3.5 μm
RB2930	30min	10min	70 - 100 nm	3 - 3.5 μm
RB2931	40min		80 - 120 nm	2.5 - 3 μm
RB2933	30min	10min	70 - 100 nm	3 - 3.5 μm

Table 4.1: List of growth runs performed at 480°C. The medium flow uses a molar flow of 4.18E-07 TMIn and 3.10E-04 TMSb. The high flow uses 5.58E-07 TMIn and 4.11E-04 TMSb. The growth time for each step is indicated as well as the resulting nanowire length and diameter.

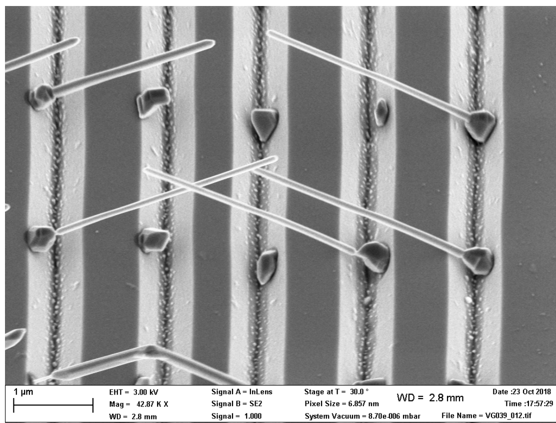
In Figure 4.15 SEM images can be seen of the four growth runs from table 4.1. RB2929 has the lowest yield of all the experiments in the series, with a lot of overgrown stems. This indicates that the high flow step with high encapsulation starts too early, after 20 minutes of medium flow. RB2930 has a higher yield, which is expected as the medium flow step is longer than before, reducing the amount of overgrown wires. However, not all the nanowires are fully encapsulated. RB2931 has a slightly lower yield with more overgrowth, but the nanowires that do grow have fully encapsulated stems. This is unexpected, as only the medium flow is used which should result in higher yield and less encapsulation than previously. A possible explanation is that the InP stems on this sample were slightly shorter resulting in quicker overgrowth. RB2933 is done to reproduce the high yield of RB2930, which is indeed observed.



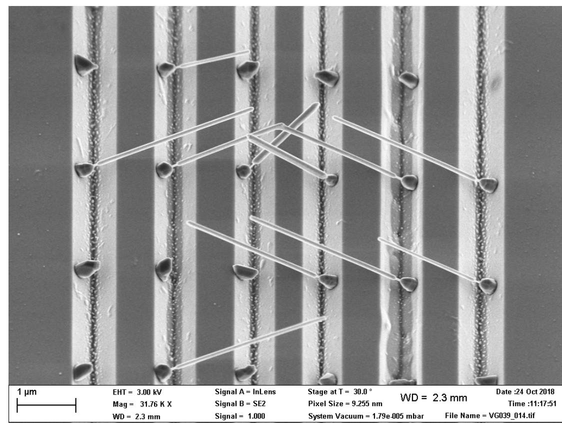
(a) RB2929.



(b) RB2930.



(c) RB2931.



(d) RB2933.

Figure 4.15: SEM images of 3x3 trenches with InSb grown on InP stems.

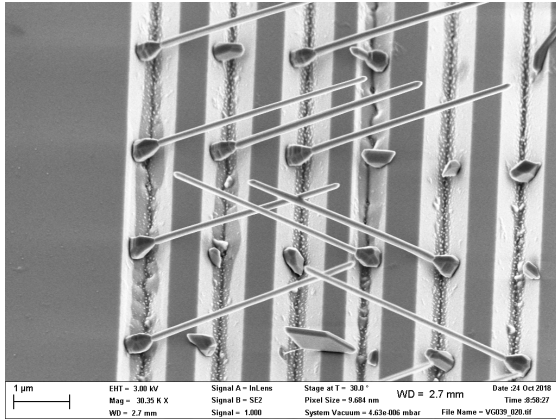
4.3.2 Three step InSb growth

With a high yield, (nearly) encapsulated InP stems and thin (70-100nm) InSb nanowires the basis for the nanowire networks is achieved. However, the nanowires need to be longer (up to 6μm) to get completed 3x3 and larger nanowire networks. To investigate the best way to increase the InSb nanowire length without increasing the diameter a flow series is done. First the nanowires are grown as in the optimized two step recipe of RB2930 (table 4.1), after which the flow is lowered for a third growth step in order to suppress radial growth. In table 4.2 the molar fractions and growth time for the third growth step is shown for the different growth runs. The resulting nanowire diameter and length is indicated as well.

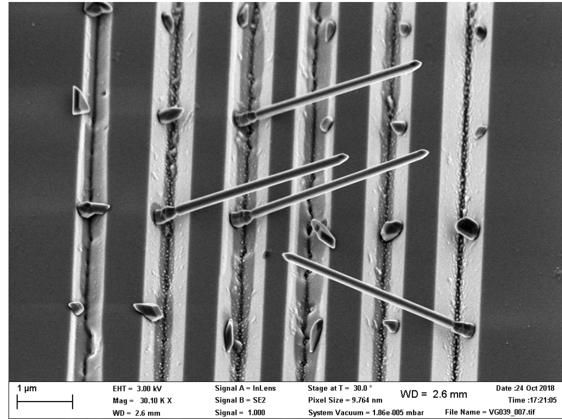
Run	TMIn flow	TMSb flow	Time (min)	Diameter	Length
RB2932	5.58E-07	4.11E-04	20	120 - 150 nm	3.5 - 3.8 μm
RB2934	4.18E-07	3.10E-04	20	120 - 150 nm	3.5 - 3.8 μm
RB2936	2.09E-07	1.55E-04	20	90 - 120 nm	3.5 - 3.8 μm
RB2937	1.12E-07	7.85E-05	20	70 - 100 nm	3.5 - 3.8 μm

Table 4.2: List of the third growth step of runs performed at 480°C. The first growth step consists of 30 minutes of 4.18E-07 TMIn and 3.10E-04 TMSb. The second growth step consists of 10 minutes of 5.58E-07 TMIn and 4.11E-04 TMSb. The TMIn and TMSb molar fractions and growth time for the third growth step are indicated as well as the resulting nanowire diameter and length.

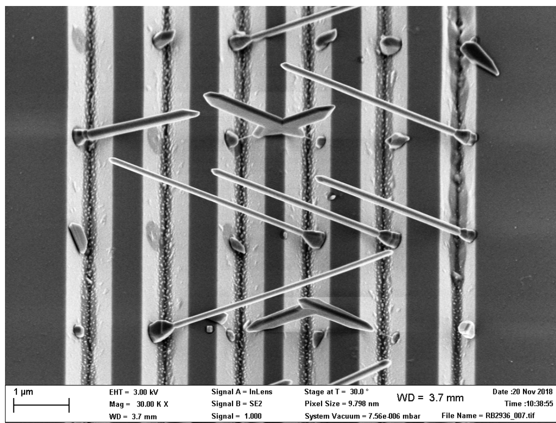
In Figure 4.16 the four growth runs from table 4.2 are shown. The yield is similar for all four, which is expected as the first 40 minutes of growth that determine whether a nanowire grows or the stem is overgrown are identical. Not only the yield, but also the nanowire length is similar with nanowires around 3.5 to 3.8μm length. The nanowire diameter follows a clear trend. The higher the flow in the third step, the more radial growth which results in a larger diameter. With the lowest flow the radial growth is strongly reduced, which is expected as the gold assisted VLS growth is less affected by the precursor flow than direct adsorption (VS).



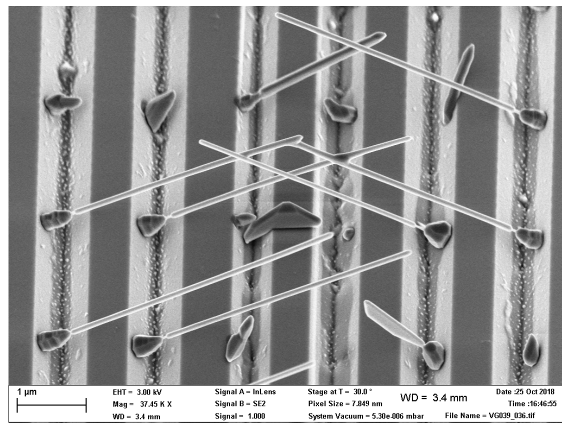
(a) RB2932.



(b) RB2934.



(c) RB2936.



(d) RB2937.

Figure 4.16: SEM images of 3x3 trenches with InSb grown on InP stems.

4.3.3 Optimizing InSb length

Now with suitable conditions found in the previous section to grow the InSb nanowires in length while largely suppressing radial growth, the next step is attempting to grow longer nanowires using longer growth times, which should lead to completed nanowire networks. The growth recipe used for these experiments is shown in table 4.3.

Precursor	Step 1	Step 2	Step 3
TMIn	4.18E-07	5.58E-07	1.12E-07
TMSb	3.10E-04	4.11E-04	7.85E-05
Time	30min	10min	variable

Table 4.3: List of the TMIn and TMSb molar fractions and growth time for each growth step. The growth temperature is 480°C.

In Figure 4.17 the third growth step is done for 50 minutes. The nanowires have a length of approximately 4μm, with a diameter of 100 to 120nm. In Figure 4.17a a nanocross (1x1 structure) is shown and in Figure 4.17b a nanohashtag (2x2 structure) is shown. The nanowires resulting from this experiment are too short for complete larger structures.

The main observation from this experiment is that although the yield of the nanowires is high, the yield of the structures is quite low. This is caused by a design flaw which can be seen in Figure 4.17c. The gold droplets are placed too close to the center of trench, which causes the nanowires to grown on only one side of the trenches. The sine function added to the droplets as explained in section 3.2.1 slightly alleviates the problem. On the majority of the sample the nanowires only grow on one of the sides of the trenches, switching periodically from side to side. However, there is a small transition region where on both sides of the trench nanowires grow, resulting in 1x1 and 2x2 nanowire networks.

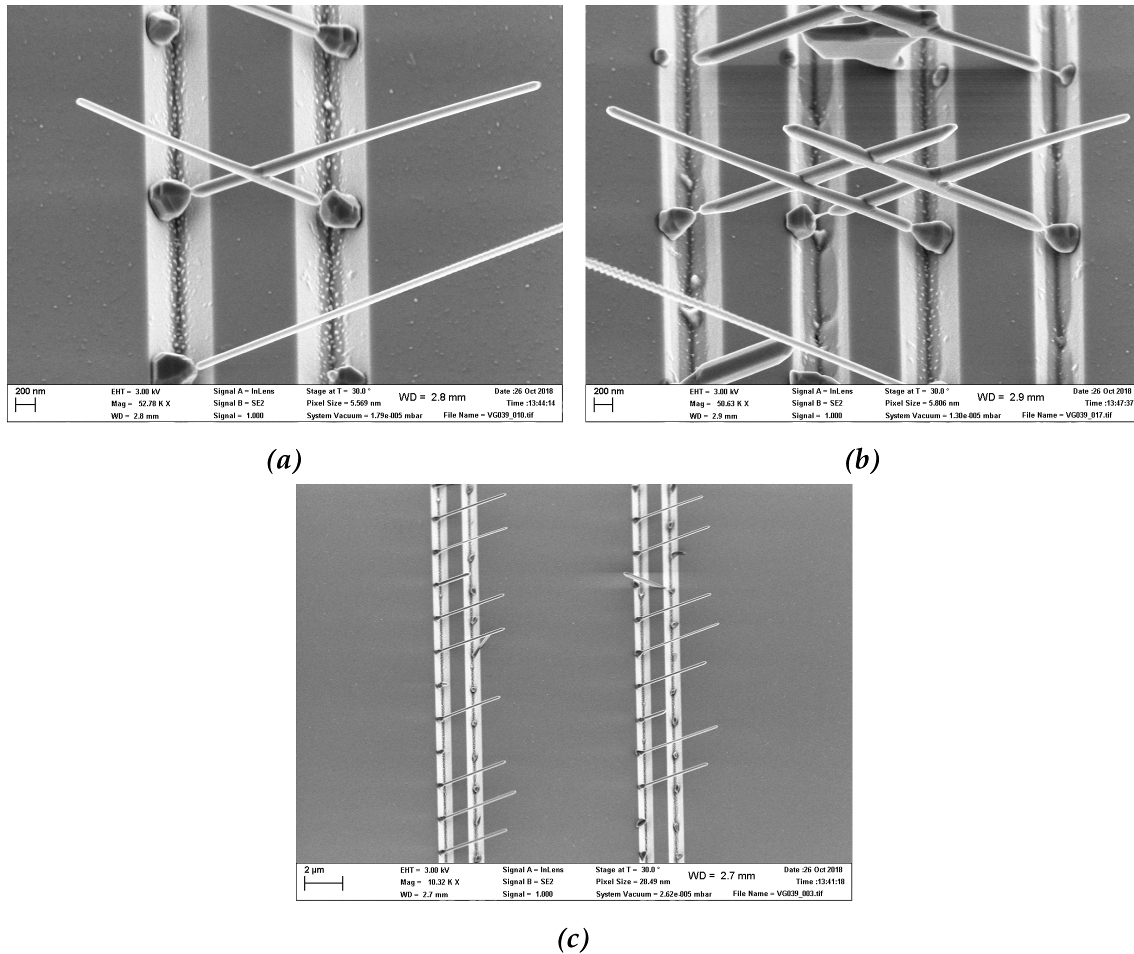
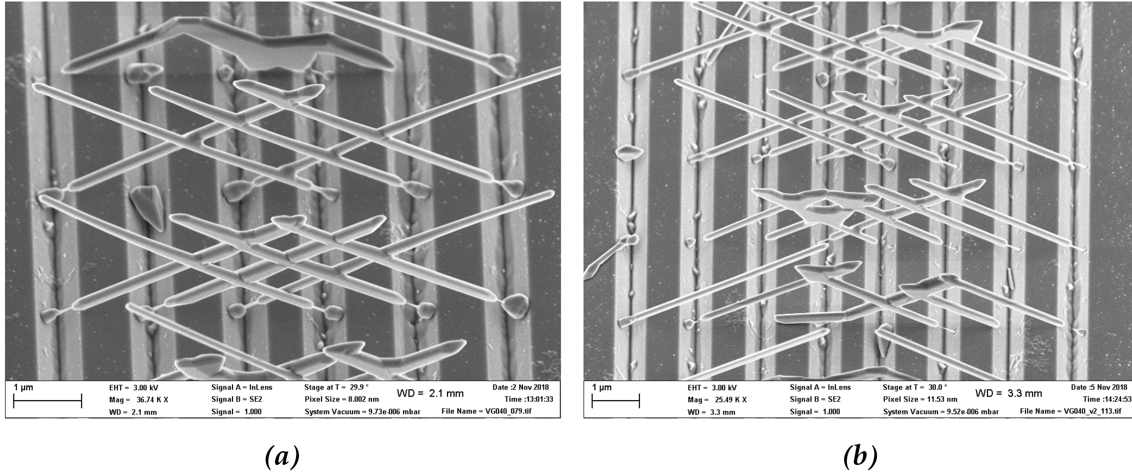


Figure 4.17: SEM images of (a) an InSb nanocross (1x1 structure) and (b) an InSb nanohashtag (2x2 structure). In (c) two nanocross designs are shown. The nanowires only grow on one side.

To get a higher yield of structures samples are fabricated with an updated design, and 500nm long InP stems are grown. Then the same growth recipe as in the previous experiment is done, with the third growth step increased to 1 hour and 50 minutes. The yield is very high, with nanowires around 5μm long, which is enough to get 3x3 structures.

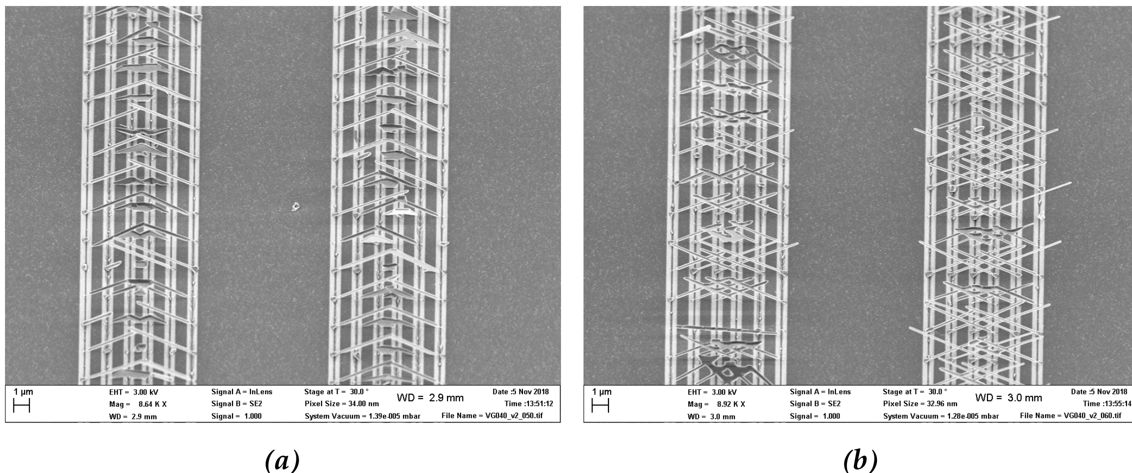
In Figure 4.18 a few of the resulting structures are shown. Figure 4.18a shows a 3x2 structure on top and a 3x3 structure on the bottom. Figure 4.18b shows an unfinished 4x3 structure.



(a) *(b)*

Figure 4.18: SEM images of (a) 3x3 and 3x2 InSb nanostructures and (b) 4x3 InSb nanostructures.

In Figure 4.19 the importance of Δy in the design can be seen. In Figure 4.19a Δy is too small, which causes the nanowires to hit each other which prevents proper junctions and can cause flake forming where the nanowires hit. Figure 4.19b shows an area where Δy is large enough to allow for proper merging behaviour. In the next section the merging behaviour is analysed more in depth.



(a) *(b)*

Figure 4.19: SEM images of (a) 3x3 and 3x2 InSb nanostructures and (b) 4x3 InSb nanostructures.

4.3.4 Merging behaviour

Besides the nanowire growth itself, another important thing is the way in which the nanowires merge. A good epitaxial connection is needed, preferably leading to a mono-crystalline structure. During the experiments two different ways of merging are observed, which can be seen in Figure 4.20. In (a) the nanowires merge properly, with a clear epitaxial connection of the nanowires. In (b) the nanowires merge as well, but there is a large amount of radial growth observed. Another observation is that the stems on merging nanowires without radial growth are fully encapsulated, but the stems of the nanowires exhibiting strong radial growth after merging are almost fully evaporated, with most structures only held up by 1 or 2 remaining stems.

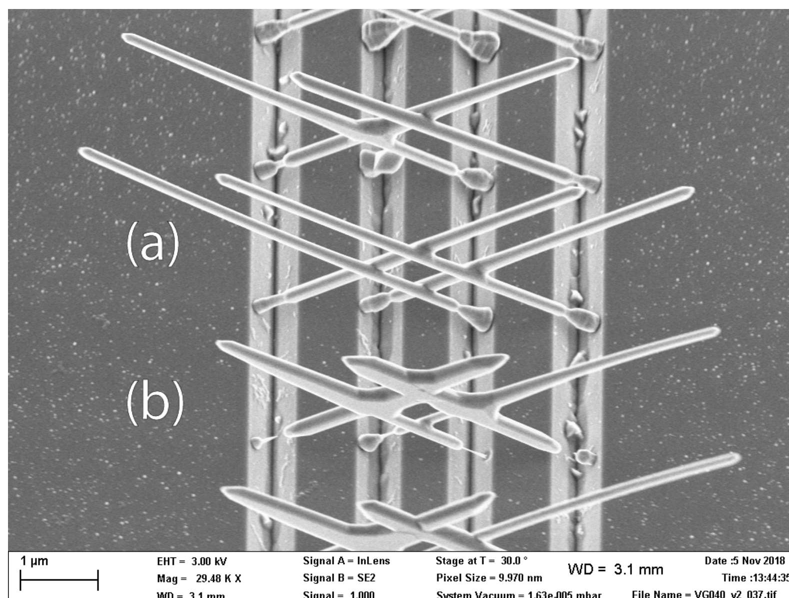
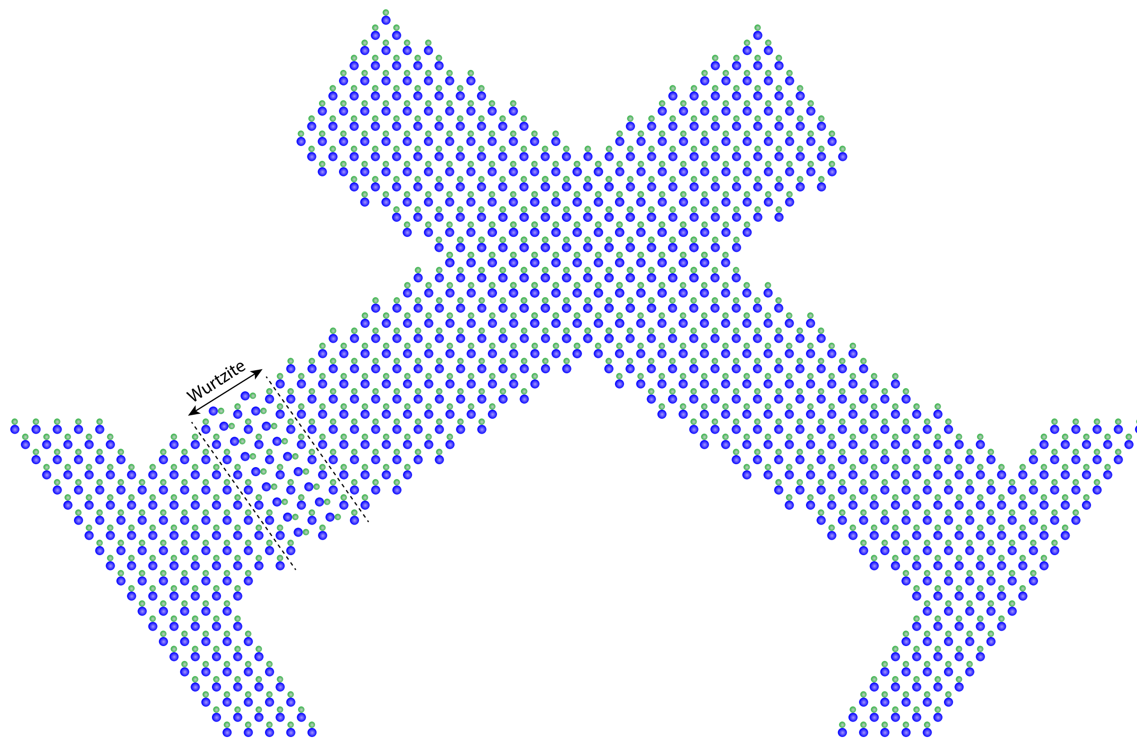


Figure 4.20: SEM image of (a) a properly merging nanowire network and (b) a merging nanowire network with strong radial growth.

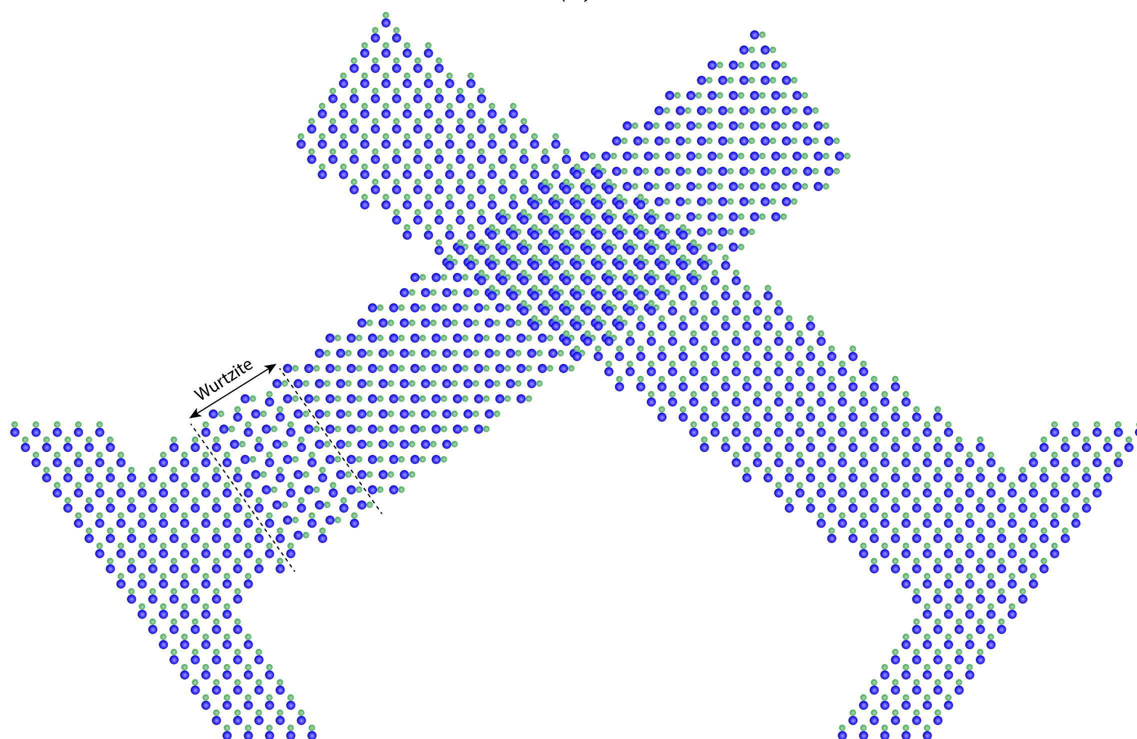
A possible explanation for the merging behaviour is the very high defect rate in InP nanowires. Depending on the amount of defects the merging nanowires can have a single crystalline or no single crystalline junction. An example of this is illustrated in Figure 4.21, where a wurtzite segment is present in the InP nanowire. If the wurtzite segment has an uneven amount of layers the zinc-blende continues in the same zinc-blende orientation as before the wurtzite segment. However, if the wurtzite segment has an even amount of layers the zinc-blende continues in a 180° rotated orientation, causing a mismatch where no single crystalline junction is possible. Not only wurtzite segments, but also twinning in the InP stems can cause this mismatch. Now due to the mismatch high energy nucleation points can

appear at the merging point, allowing radial growth over the junction. This radial growth is in a competitive regime with the stem encapsulation, which slows the encapsulation process and causes evaporated InP stems.

A second effect that can contribute to radial growth around the junctions is the nanowire offset (Δy in Figure 2.13). If the nanowires merge too close to each other, the gold droplet can be distorted, partially merging with the gold droplet of the other nanowire. In contrast, if the nanowire offset Δy is large enough, the gold droplets cannot merge and the nanowires can continue growing like normal after merging.



(a)



(b)

Figure 4.21: Schematic illustration of two possible ways of merging nanowires.

Chapter 5

Conclusion

The main goal of this thesis is to investigate whether it is possible to fabricate InSb nanowire networks suitable for Majorana physics and building a topological qubit. A platform is used in which (111)B facets are etched into an InP(100) substrate, after which gold catalysed VLS nanowires are grown on the (111)B facets. Previously this platform was limited by InP stem evaporation, and only allowed for nanohashtags (2x2) to be grown. However, to realise a single qubit device larger nanowire networks are required.

First an attempt is made to fully eliminate the InP stems by growing stemless InSb nanowires. The optimum growth temperature is investigated, and a large range of precursor flows and V/III ratios are used. At low precursor flows the nanowires crawl up the trenches and grow in the 'wrong' (111)B direction, which prevents the formation of nanowire networks. This might be caused by P evaporating from the substrate, which can prevent Sb from entering the gold catalyst. This can allow the gold to crawl along the trenches. At high precursor flows flakes are formed instead of nanowires. The high supersaturation in the gold catalyst enlarges the catalyst droplet, which allows the formation of high energy nucleation points. Catalyst free growth occurs in the (110) direction on these nucleation points, which facilitates the formation of flakes.

Then a growth scheme is developed which uses short InP stems on which InSb nanowires are grown. The goal is to encapsulate the InP stem with InSb, which prevents further evaporation. The InP stem length is observed to be critical for the subsequent InSb growth, as too short stems are encapsulated too quickly (overgrown) and prevent further InSb growth. With InP stems of 500nm initial overgrowth is strongly reduced compared to the 200 and 300nm long stems. A three step process is then developed to allow long and thin InSb nanowires to

grow while simultaneously encapsulating the InP stems. The process consists of a medium flow step to allow both InSb nanowire growth and relatively slow encapsulation. The second step uses a higher flow which encapsulates the stem faster. Once the stems are (almost) fully encapsulated the third step is started, which uses a low flow to reduce further radial growth of the InSb nanowires.

It is then demonstrated that nanowire networks up to 3x3 nanowires can be consistently fabricated, which are the required structures for a single qubit device. However, the merging points are not always perfect. Often radial growth is observed around the junctions, which might be caused by mismatch in the zinc-blende crystal orientation of the nanowires. This causes high energy defects which allow radial growth around the junction. A second contributing factor for the radial growth is that the nanowire offset Δy can be too small. In that case the gold droplets can make contact and partially merge, which can promote radial growth.

5.0.1 Outlook

Now that it is established that the semiconductor nanowire networks required for a single qubit device can be reliably grown, the first step is to demonstrate braiding. Then the next step is to fabricate and investigate a single qubit device using the 3x3 nanowire networks.

For the nanowire networks based on the InSb on InP stem growth scheme, future improvements should focus on decreasing the nanowire diameter. It will be very hard to increase the size of the structures beyond 3x3, mainly due to the radial growth on the junctions induced by mismatch of the nanowire zinc-blende structure. However, further attempts can be made to grow stemless InSb nanowires, which should suffer no such problem. One could even attempt to use the crawling nanowires to grow nanowire networks.

Bibliography

[1] R. R. Schaller, "Moore's law: past, present and future," *IEEE Spectrum*, vol. 34, pp. 52–59, June 1997.

[2] I. L. Markov, "Limits on fundamental limits to computation," *Nature*, vol. 512, pp. 147–154, Aug. 2014.

[3] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines," *Journal of Statistical Physics*, vol. 22, pp. 563–591, May 1980.

[4] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, pp. 467–488, June 1982.

[5] J. Preskill, "Quantum computing and the entanglement frontier," *arXiv:1203.5813 [cond-mat, physics:quant-ph]*, Mar. 2012. arXiv: 1203.5813.

[6] K. L. Brown, W. J. Munro, and V. M. Kendon, "Using quantum computers for quantum simulation," *Entropy*, vol. 12, no. 11, pp. 2268–2307, 2010.

[7] E. Majorana, "Theory of the Symmetry of Electrons and Positrons," *Nuovo Cim*, vol. 14, no. 171, p. 50, 1937.

[8] S. D. Sarma, M. Freedman, and C. Nayak, "Majorana zero modes and topological quantum computation," *npj Quantum Information*, vol. 1, p. 15001, Oct. 2015.

[9] A. Kitaev, "Unpaired Majorana fermions in quantum wires," *Physics-Uspekhi*, vol. 44, pp. 131–136, Oct. 2001. arXiv: cond-mat/0010440.

[10] Y. Oreg, G. Refael, and F. von Oppen, "Helical liquids and Majorana bound states in quantum wires," *Physical review letters*, vol. 105, no. 17, p. 177002, 2010.

[11] M. Leijnse and K. Flensberg, "Introduction to topological superconductivity and Majorana fermions," *Semiconductor Science and Technology*, vol. 27, no. 12, p. 124003, 2012.

[12] V. Mourik, K. Zuo, S. M. Frolov, S. R. Plissard, E. P. a. M. Bakkers, and L. P. Kouwenhoven, "Signatures of Majorana Fermions in Hybrid Superconductor-Semiconductor Nanowire Devices," *Science*, vol. 336, pp. 1003–1007, May 2012.

[13] H. Zhang, C.-X. Liu, S. Gazibegovic, D. Xu, J. A. Logan, G. Wang, N. Van Loo, J. D. Bommer, M. W. De Moor, and D. Car, "Quantized Majorana conductance," *Nature*, vol. 556, no. 7699, p. 74, 2018.

[14] V. Lahtinen and J. K. Pachos, "A Short Introduction to Topological Quantum Computation," *SciPost Physics*, vol. 3, Sept. 2017. arXiv: 1705.04103.

[15] J. Alicea, Y. Oreg, G. Refael, F. v. Oppen, and M. P. A. Fisher, "Non-Abelian statistics and topological quantum information processing in 1d wire networks," *Nature Physics*, vol. 7, pp. 412–417, May 2011.

[16] T. Karzig, C. Knapp, R. M. Lutchyn, P. Bonderson, M. B. Hastings, C. Nayak, J. Alicea, K. Flensberg, S. Plugge, and Y. Oreg, "Scalable designs for quasiparticle-poisoning-protected topological quantum computation with Majorana zero modes," *Physical Review B*, vol. 95, no. 23, p. 235305, 2017.

[17] S. Gazibegovic, D. Car, H. Zhang, S. C. Balk, J. A. Logan, M. W. de Moor, M. C. Cassidy, R. Schmits, D. Xu, and G. Wang, "Epitaxy of advanced nanowire quantum devices," *Nature*, vol. 548, no. 7668, p. 434, 2017.

[18] S. Plugge, A. Rasmussen, R. Egger, and K. Flensberg, "Majorana box qubits," *New Journal of Physics*, vol. 19, p. 012001, Jan. 2017.

[19] K. Tanabe, D. Guimard, D. Bordel, and Y. Arakawa, "High-efficiency InAs/-GaAs quantum dot solar cells by metalorganic chemical vapor deposition," *Applied Physics Letters*, vol. 100, p. 193905, May 2012.

[20] V. M. Ustinov, N. A. Maleev, A. E. Zhukov, A. R. Kovsh, A. Y. Egorov, A. V. Lunev, B. V. Volovik, I. L. Krestnikov, Y. G. Musikhin, N. A. Bert, P. S. Kop'ev, Z. I. Alferov, N. N. Ledentsov, and D. Bimberg, "InAs/InGaAs quantum dot structures on GaAs substrates emitting at 1.3 μm," *Applied Physics Letters*, vol. 74, pp. 2815–2817, May 1999.

[21] N. Panev, A. I. Persson, N. Sköld, and L. Samuelson, "Sharp exciton emission from single InAs quantum dots in GaAs nanowires," *Applied Physics Letters*, vol. 83, pp. 2238–2240, Sept. 2003.

[22] C. Downs and T. Vandervelde, "Progress in Infrared Photodetectors Since 2000," *Sensors (Basel, Switzerland)*, vol. 13, pp. 5054–5098, Apr. 2013.

[23] R. S. Wagner and W. C. Ellis, "Vapor-liquid-solid mechanism of single crystal growth," *Applied Physics Letters*, vol. 4, no. 5, pp. 89–90, 1964.

[24] J. Wang, *Controlling nanowire growth direction*. PhD Thesis, Eindhoven University of Technology, 2014.

[25] D. Car, J. Wang, M. A. Verheijen, E. P. A. M. Bakkers, and S. R. Plissard, "Rationally Designed Single-Crystalline Nanowire Networks," *Advanced Materials*, vol. 26, pp. 4875–4879, July 2014.

[26] A. Pimpin and W. Srituravanich, "Review on micro-and nanolithography techniques and their applications," *Engineering Journal*, vol. 16, no. 1, pp. 37–56, 2012.

[27] C. Cardinaud, M.-C. Peignon, and P.-Y. Tessier, "Plasma etching: principles, mechanisms, application to micro- and nano-technologies," *Applied Surface Science*, vol. 164, pp. 72–83, Sept. 2000.

[28] D. Bour, Z. Yang, and C. Chua, "Simple technique for measuring the filled volume of liquid or solid CVD precursor chemicals in bubblers," *Journal of Crystal Growth*, vol. 310, pp. 2673–2677, May 2008.

[29] J. L. Zilko, "Metal organic chemical vapor deposition: technology and equipment," in *Handbook of Thin Film Deposition Processes and Techniques (Second Edition)*, pp. 151–203, Elsevier, 2001.

[30] Carl Zeiss Microscopy GmbH, "Zeiss sigma family product information." `https://www.zeiss.com/microscopy`. Accessed: 2018-06-27.

[31] M. Shafa, S. Akbar, L. Gao, M. Fakhar-e Alam, and Z. M. Wang, "Indium Antimonide Nanowires: Synthesis and Properties," *Nanoscale Research Letters*, vol. 11, Dec. 2016.

[32] M. Hocevar, G. Immink, M. Verheijen, N. Akopian, V. Zwiller, L. Kouwenhoven, and E. Bakkers, "Growth and optical properties of axial hybrid III–V/silicon nanowires," *Nature Communications*, vol. 3, p. 1266, Dec. 2012.

[33] S. Gazibegovic and G. Badawy, "Bottom-up grown two-dimensional InSb nanostructures," *Manuscript in Preparation*.

Appendix A

Growth experiments

Run	TMIn flow	TMSb flow	V/III ratio
RB2768	1.95E-07	1.23E-04	6.31E+02
RB2787	1.95E-07	3.10E-04	1.59E+03
RB2788	1.95E-07	5.23E-04	2.68E+03
RB2789	1.95E-07	1.12E-03	5.74E+03
RB2792	1.95E-07	2.24E-03	1.15E+04
RB2793	4.18E-07	1.12E-03	2.68E+03
RB2794	1.12E-07	1.12E-03	1.00E+04
RB2795	4.18E-08	1.12E-03	2.68E+04

Table A.1: List of growth runs performed for one hour at 455°C with the TMIn and TMSb molar fraction and V/III ratio indicated.

Run	TMIn flow	TMSb flow	V/III ratio
RB2796	1.95E-07	1.12E-03	5.74E+03
RB2797	1.95E-07	5.23E-04	2.68E+03
RB2798	1.95E-07	2.24E-03	1.15E+04
RB2799	1.12E-07	2.99E-04	2.67E+03
RB2800	1.12E-07	6.35E-04	5.67E+03
RB2801	1.12E-07	1.27E-03	1.13E+04

Table A.2: List of growth runs performed for 3 hours at 455°C with the TMIn and TMSb molar fraction and V/III ratio indicated.

Appendix B

Design software code

Listing B.1: GUI.py

```
import wx
import Functions
import math
from dxfwrite import DXFEngine as dxf
import os
import Globals
import configparser
import matplotlib.pyplot as plt

class helpDialog(wx.Frame):
    def __init__(self,parent,title):
        super(helpDialog, self).__init__(parent, title=title, size=(508, 454))
        self.InitUI()

    def InitUI(self):
        panelhelp = wx.Panel(self,-1)
        png = wx.Image('images\HelpFigure.png', wx.BITMAP_TYPE_ANY).ConvertToBitmap()
        wx.StaticBitmap(self, -1, png, (0, 0), (png.GetWidth(), png.GetHeight()))
        helplabel1 = wx.StaticText(self,-1,pos=(20,300),size=(450,200),style=wx.TE_MULTILINE)
        helplabel1.SetLabel("a and b parameters are indicated here. To make a new drawing restart the program."
                            "\n Once you save a .txt file with all your settings is saved in the same folder"
                            "\n with the same name. Some more tweakable parameters can be found in config.ini"
                            "\n More help may or may not (most probably not"
                            ") be added in the future. ")


class EBLWriter(wx.Frame):

    def __init__(self,parent,title):
        super(EBLWriter,self).__init__(parent,title=title, size=(700,440))

        self.InitUI()
        self.Centre()
        self.Show()
```

```
        self.startDrawing()
        Functions.Configparser()

        self.defaultValues()

        icon = wx.Icon('images\Icon1.ico', wx.BITMAP_TYPE_ICO)
        self.SetIcon(icon)


    def InitUI(self):

        panel = wx.Panel(self,-1)

        menubar = wx.MenuBar()

        # Make menu 1
        FirstMenu = wx.Menu()
        fitem1 = FirstMenu.Append(wx.ID_ABOUT,'About')
        fitem3 = FirstMenu.Append(wx.ID_HELP,'Help')
        fitem2 = FirstMenu.Append(wx.ID_EXIT, 'Quit', 'Quit application')

        self.Bind(wx.EVT_MENU, self.OnAbout,fitem1)
        self.Bind(wx.EVT_MENU, self.OnHelp,fitem3)
        self.Bind(wx.EVT_MENU, self.OnQuit,fitem2)

        # Make menu 2
        SecondMenu = wx.Menu()
        sitem3 = SecondMenu.Append(wx.ID_NEW,'New drawing')
        sitem1 = SecondMenu.Append(wx.ID_SAVE,'Save')
        sitem2 = SecondMenu.Append(wx.ID_SAVEAS,'Save as')

        self.Bind(wx.EVT_MENU, self.SaveButtonFunc,sitem1)
        self.Bind(wx.EVT_MENU, self.saveAsFunc,sitem2)
        self.Bind(wx.EVT_MENU, self.newDrawing,sitem3)

        # Add all menus to menubar
        menubar.Append(FirstMenu, '&Menu')
        menubar.Append(SecondMenu, '&File')


        self.SetMenuBar(menubar)

        #Make all buttons here
        hashtag1lbl = wx.StaticText(panel,-1,pos=(10,0),style=wx.ALIGN_LEFT)
        hashtag1lbl.SetLabel("Hashtag size (ex. a x b)")
        # without the self the OKButtonFunc cannot read it
        self.dlghashtag1size = wx.TextCtrl(panel, pos=(200,0), size=(40,20),value="3")

        # without the self the OKButtonFunc cannot read it
        self.dlghashtag2size = wx.TextCtrl(panel, pos=(250,0), size=(40,20),value="3")

        aparamlbl = wx.StaticText(panel,-1,pos=(10,20),style=wx.ALIGN_LEFT)
        aparamlbl.SetLabel("a parameter [nm]")
        self.dlgaparam = wx.TextCtrl(panel, pos=(200,20), size=(40,20),value="800")

        bparamlbl = wx.StaticText(panel,-1,pos=(10,40),style=wx.ALIGN_LEFT)
```

```
bparamlbl.SetLabel("b parameter [nm]")
self.dlgbparam = wx.TextCtrl(panel, pos=(200,40), size=(40,20),value="800")

stemlengthlbl = wx.StaticText(panel,-1,pos=(10,60),style=wx.ALIGN_LEFT)
stemlengthlbl.SetLabel("Stem length [nm]")
self.dlgstemlength = wx.TextCtrl(panel, pos=(200,60), size=(40,20),value="500")

wirelengthlbl = wx.StaticText(panel,-1,pos=(10,80),style=wx.ALIGN_LEFT)
wirelengthlbl.SetLabel("wire length [nm]")
self.dlgwirelength = wx.TextCtrl(panel, pos=(200,80), size=(40,20),value="6000")

wirediameterlbl = wx.StaticText(panel,-1,pos=(10,100),style=wx.ALIGN_LEFT)
wirediameterlbl.SetLabel("wire diameter [nm]")
wirediameterlblmin = wx.StaticText(panel,-1,pos=(170,100),style=wx.ALIGN_LEFT)
wirediameterlblmin.SetLabel("Min")
self.dlgwirediametermin = wx.TextCtrl(panel, pos=(200,100), size=(40,20),value="20")
wirediameterlblmax = wx.StaticText(panel,-1,pos=(270,100),style=wx.ALIGN_LEFT)
wirediameterlblmax.SetLabel("Max")
self.dlgwirediametermax = wx.TextCtrl(panel, pos=(300,100), size=(40,20),value="50")
wirediameterlblstep = wx.StaticText(panel,-1,pos=(370,100),style=wx.ALIGN_LEFT)
wirediameterlblstep.SetLabel("Step")
self.dlgwirediameterstep = wx.TextCtrl(panel, pos=(400,100), size=(40,20),value="10")

wirepitchlbl = wx.StaticText(panel,-1,pos=(10,120),style=wx.ALIGN_LEFT)
wirepitchlbl.SetLabel("wire pitch [nm]")
self.dlgwirepitch = wx.TextCtrl(panel, pos=(200,120), size=(40,20),value="2000")

trenchlengthlbl = wx.StaticText(panel,-1,pos=(10,140),style=wx.ALIGN_LEFT)
trenchlengthlbl.SetLabel("trench length [um]")
self.dlgtrenchlength = wx.TextCtrl(panel, pos=(200,140), size=(40,20),value="200")

dotshiftlbl = wx.StaticText(panel,-1,pos=(10,160),style=wx.ALIGN_LEFT)
dotshiftlbl.SetLabel("Dot shift [nm]")
dotshiftlblmin = wx.StaticText(panel,-1,pos=(170,160),style=wx.ALIGN_LEFT)
dotshiftlblmin.SetLabel("Min")
self.dlgdotshiftmin = wx.TextCtrl(panel, pos=(200,160), size=(40,20),value="0")
dotshiftlblmax = wx.StaticText(panel,-1,pos=(270,160),style=wx.ALIGN_LEFT)
dotshiftlblmax.SetLabel("Max")
self.dlgdotshiftmax = wx.TextCtrl(panel, pos=(300,160), size=(40,20),value="150")

rectanglewidthlbl = wx.StaticText(panel,-1,pos=(10,180),style=wx.ALIGN_LEFT)
rectanglewidthlbl.SetLabel("rectangle width [nm]")
self.dlgrectanglewidth = wx.TextCtrl(panel, pos=(250,180), size=(40,20),value="180")

trenchdepthlbl = wx.StaticText(panel,-1,pos=(10,200),style=wx.ALIGN_LEFT)
trenchdepthlbl.SetLabel("trench depth [nm]")
self.dlgtrenchdepth = wx.TextCtrl(panel, pos=(250,200), size=(40,20),value="500")

trenchseparationlbl = wx.StaticText(panel,-1,pos=(10,220),style=wx.ALIGN_LEFT)
trenchseparationlbl.SetLabel("trench seperation in x direction [um]")
self.dlgtrenchseparation = wx.TextCtrl(panel, pos=(250,220), size=(40,20),value="5")

self.trenchCounter = wx.StaticText(panel,-1,pos=(10,320))
self.lineCounter = wx.StaticText(panel,-1,pos=(10,340))
self.blocksCounter = wx.StaticText(panel,-1,pos=(10,360))
```

```
        #Checkbox for shadowing wires
        self.cb1 = wx.CheckBox(panel,label = 'Single shadow',pos=(300,1))
        self.cb2 = wx.CheckBox(panel, label='Double shadow', pos=(300, 21))
        self.Bind(wx.EVT_CHECKBOX, self.singleshadowFunc,self.cb1)
        self.Bind(wx.EVT_CHECKBOX, self.doubleshadowFunc,self.cb2)

        OKbutton = wx.Button(panel, label='Save drawing', size = (200,60), pos=(430,250))
        self.Bind(wx.EVT_BUTTON, self.SaveButtonFunc, OKbutton)

        Plotbutton = wx.Button(panel, label='Plot 2D wires', size = (200,30), pos=(10,250))
        self.Bind(wx.EVT_BUTTON, self.PlotbuttonFunc, Plotbutton)

        DXFWritebutton = wx.Button(panel, label='Write block with current settings', size = (200,30), pos=(220,250))
        self.Bind(wx.EVT_BUTTON, self.DXFWritebuttonFunc, DXFWritebutton)

        NewLinebutton = wx.Button(panel, label='Start on new line', size = (200,30), pos=(220,280))
        self.Bind(wx.EVT_BUTTON, self.NewLinebuttonFunc, NewLinebutton)

        NewBlockbutton = wx.Button(panel, label='Start new block', size = (200,30), pos=(220,310))
        self.Bind(wx.EVT_BUTTON, self.NewBlockbuttonFunc, NewBlockbutton)

        EstimatexSize = wx.Button(panel, label='Estimate # of blocks per line', size = (200,30), pos=(10,280))
        self.Bind(wx.EVT_BUTTON, self.EstimatexSizeFunc, EstimatexSize)

        multipleTrenchbutton = wx.Button(panel, label='Write range of trenches', size = (200,30), pos=(300,50))
        self.Bind(wx.EVT_BUTTON, self.multipleTrenchFunc, multipleTrenchbutton)

    #Menu functions
    def OnQuit(self,event):
        self.Close()

    def OnAbout(self,event):
        msg = 'Made for TU/e by Sander Schellingerhout'
        dlg = wx.MessageDialog(parent=None,message=msg,caption='About')
        dlg.ShowModal()
        dlg.Destroy()

    def OnHelp(self,event):
        secondWindow = helpDialog(None, title='Help')
        secondWindow.Show()

    def singleshadowFunc(self,event):
        if self.cb1.GetValue() == 1 and self.cb2.GetValue() == 1:
            self.cb2.SetValue(0)

    def doubleshadowFunc(self,event):
        if self.cb1.GetValue() == 1 and self.cb2.GetValue() == 1:
            self.cb1.SetValue(0)

    #functions
    def startDrawing(self):
        self.drawing = dxf.drawing()
        self.drawing.add_layer('Rectangles', color=2)
        self.drawing.add_layer('Dots', color=4)
        self.drawing.add_layer('TrenchSize', color=5)
        self.drawing.add_layer('TextMarkers', color=6)
```

```
        self.drawing.header['$INSUNITS'] = 12 # set scale to nanometers

        self.SaveLocation = ""
        self.writeButtonClicks = 0
        self.x=0
        self.y=0
        self.lineNumber = 1
        Globals.xBlockNr = 0
        Globals.blockfullwarning = 0
        Globals.linefullwarning = 0

        self.trenchCounter.SetLabel('Number of trenches written: %s' % self.writeButtonClicks)
        self.lineCounter.SetLabel('Current line: %s' % self.lineNumber)
        self.blocksCounter.SetLabel('Current block: %s' % int(Globals.xBlockNr+1))

    def newDrawing(self,event):
        newDrawingmsg = 'Everything that is not saved will be gone. Continue?'
        newDrawingprompt = wx.MessageBox(newDrawingmsg,'New drawing',wx.YES_NO)
        if newDrawingprompt == wx.YES:
            self.startDrawing()
            Globals.dataLogFile = ""
        else:
            return()

    def saveLocationFunc(self):
        SaveDir = wx.FileDialog(self, "Select file location", wildcard="dxf files (*.dxf)|*.dxf",
                                style=wx.FD_SAVE | wx.FD_OVERWRITE_PROMPT)
        if SaveDir.ShowModal() == wx.ID_OK:
            self.SaveLocation = SaveDir.GetPath()
        else:
            return()
        SaveDir.Destroy()

    def saveAsFunc(self,event):
        self.saveLocationFunc()
        self.SaveButtonFunc(wx.EVT_BUTTON)

    #button functions
    def OKButtonFunc(self,event):
        singleShadow = self.cb1.GetValue()
        doubleShadow = self.cb2.GetValue()
        if doubleShadow == True:
            Globals.shadowing = 2
        elif singleShadow == True:
            Globals.shadowing = 1
        else:
            Globals.shadowing = 0

        hashtagSize = [int(self.dlghashtag1size.GetValue()),int(self.dlghashtag2size.GetValue())] #hashtag size
        aParam = float(self.dlgaparam.GetValue()) #a parameter
        bParam = float(self.dlgbparam.GetValue()) # b paramter
        stemLength = float(self.dlgstemlength.GetValue()) # stem length
        wireLength = float(self.dlgwirelength.GetValue()) #wire length
        wireDiameter = [float(self.dlgwirediametermin.GetValue()),float(self.dlgwirediametermax.GetValue()),
                        float(self.dlgwirediameterstep.GetValue())] #wire diamter
        wirePitch = float(self.dlgwirepitch.GetValue()) #wire pitch
```

```
        trenchLength = 1000*float(self.dlgtrenchlength.GetValue()) #trench length
        dotShift = [float(self.dlgdotshiftmin.GetValue()),float(self.dlgdotshiftmax.GetValue())] #dot shift
        rectangleWidth = float(self.dlgrectanglewidth.GetValue()) #rectangle width
        trenchDepth = float(self.dlgtrenchdepth.GetValue()) # trench depth
        trenchSeperation = 1000*float(self.dlgtrenchseparation.GetValue()) # seperation between trenches
# numIdenticalTrenches = int(self.dlgIdenticalTrenches.GetValue()) # number of identical trenches
        numIdenticalTrenches = 0

        angle = math.radians(35.3) # in degrees; math class uses radians
        trenchWidth = 2 * trenchDepth * math.tan(angle)

        cParam = (2 * stemLength + wireLength - (hashtagSize[0] - 1) * aParam - (hashtagSize[1] - 1) * bParam) * \
                math.cos(angle) - 2 * trenchWidth / 4

        s1 = (aParam + wireDiameter[1]) / math.sin(angle)
        s2 = (bParam + wireDiameter[1]) / math.sin(angle)

        #set wirelength to add the stem for future calculations
        wireLength += stemLength

        return(hashtagSize,aParam,bParam,cParam,s1,s2,wireLength,wireDiameter,wirePitch,trenchLength,
              dotShift,rectangleWidth,trenchDepth,angle,trenchWidth,trenchSeperation,numIdenticalTrenches,
                  ↪ stemLength,
              trenchSeperation)

    def PlotbuttonFunc(self,event):
        hashtagSize, aParam, bParam, cParam, s1, s2, wireLength, wireDiameter, wirePitch, trenchLength,\
        dotShift, rectangleWidth, trenchDepth, angle, trenchWidth, trenchSeperation, numIdenticalTrenches, \
        stemLength,trenchSeperation = \
            self.OKButtonFunc(wx.EVT_BUTTON) #press OKButton to retrieve data

        Functions.plot_2D(hashtagSize,cParam,s1,s2,angle,wireLength)

    def DXFWritebuttonFunc(self,event):


        hashtagSize, aParam, bParam, cParam, s1, s2, wireLength, wireDiameter, wirePitch, trenchLength,\
        dotShift, rectangleWidth, trenchDepth, angle, trenchWidth, trenchSeperation, numIdenticalTrenches, \
        stemLength,trenchSeperation = \
            self.OKButtonFunc(wx.EVT_BUTTON) #press OKButton to retrieve data

        if cParam < trenchWidth:
            newDrawingprompt = wx.MessageBox('WARNING: The seperation between the inner most trenches is too
                ↪ small, '
                                        'overlap or other issues might occur. a: %s, b: %s. Continue?'
                                        %(aParam,bParam), 'WARNING',wx.YES_NO|wx.ICON_EXCLAMATION)
            if newDrawingprompt != wx.YES:
                Globals.linefullwarning = 1
                return()

        xBlockSize = Functions.DrawMultipleDiameterTrench(self.x,self.y,self.drawing,hashtagSize,aParam,bParam,
            ↪ cParam,
                                                    s1,s2,wireLength,
                                        wireDiameter,wirePitch,trenchLength,dotShift,rectangleWidth,
                                        angle,trenchWidth,stemLength,trenchSeperation,self.writeButtonClicks)
```

```
        self.writeButtonClicks += 1
        self.x += xBlockSize + trenchSeperation*2

        trenchCountermsg = 'Number of trenches written: %s' % int(self.writeButtonClicks)
        self.trenchCounter.SetLabel(trenchCountermsg)

        if self.y+2*trenchLength+Globals.y_seperation> Globals.block_size and self.x +xBlockSize \
                > (Globals.xBlockNr+1)*Globals.block_size and Globals.blockfullwarning == 0:
            wx.MessageBox('The next line will most likely not fit in the 500x500um square, '
                          'it is recommended to start in a new block.','Box full',wx.OK
                          | wx.ICON_INFORMATION)
            Globals.blockfullwarning = 1

        if self.x +xBlockSize > (Globals.xBlockNr+1)*Globals.block_size:
            newLineQuestion = wx.MessageBox('This line is full, continue on next? If you dont the EBL will have to '
                                            'move the stage. If you press yes while writing a range the writing will'
                                            ' continue.','Row full',wx.YES_NO
                          | wx.ICON_INFORMATION)
            if newLineQuestion == wx.YES:
                self.x = 0 + Globals.xBlockNr * Globals.block_distance
                self.y+= trenchLength+Globals.y_seperation
                self.lineNumber += 1
                self.lineCounter.SetLabel('Current line: %s' % self.lineNumber)
            else:
                Globals.linefullwarning = 1
                return()

    def multipleTrenchFunc(self,event):
        Globals.linefullwarning = 0

        dlgChoice = wx.SingleChoiceDialog(
                self, "Which variable to vary?", 'The Caption',["a range", "b range","symmetrical a and b"],
            wx.CHOICEDLG_STYLE)

        if dlgChoice.ShowModal() == wx.ID_OK:
            multipleSelect = dlgChoice.GetStringSelection()
        else:
            return()

        dlgChoice.Destroy()

        dlgChoice2 = wx.TextEntryDialog(self, 'Enter min,max,step. Put max to 0 if you want to fill the line.',
            'Range writer', '')

        if dlgChoice2.ShowModal() == wx.ID_OK:
            multipleChoice = dlgChoice2.GetValue()
        else:
            return()

        dlgChoice2.Destroy()

#        multipleValues = map(int,multipleChoice.split(","))
        multipleValues = multipleChoice.split(",")
        multipleMin = int(multipleValues[0])
        multipleMax = int(multipleValues[1])
        multipleStep = int(multipleValues[2])
```

```
        if multipleMax == 0:

            #max value is large to keep running until line is full
            for i in range(multipleMin,1000000000,multipleStep):
                if Globals.linefullwarning == 1:
                    break
                else:
                    print(i)
                    if multipleSelect == 'a range':
                        self.dlgaparam.SetValue('%s' % i)
                    elif multipleSelect == 'b range':
                        self.dlgbparam.SetValue('%s' % i)
                    elif multipleSelect == 'symmetrical a and b':
                        self.dlgaparam.SetValue('%s' % i)
                        self.dlgbparam.SetValue('%s' % i)

                    self.DXFWritebuttonFunc(wx.EVT_BUTTON)

        # +1 is to fix it stopping too early
        else:
            for i in range(multipleMin, multipleMax + 1, multipleStep):
                print(i)
                if multipleSelect == 'a range':
                    self.dlgaparam.SetValue('%s' % i)
                elif multipleSelect == 'b range':
                    self.dlgbparam.SetValue('%s' % i)
                elif multipleSelect == 'symmetrical a and b':
                    self.dlgaparam.SetValue('%s' % i)
                    self.dlgbparam.SetValue('%s' % i)
                self.DXFWritebuttonFunc(wx.EVT_BUTTON)

    def NewLinebuttonFunc(self,event):

        hashtagSize, aParam, bParam, cParam, s1, s2, wireLength, wireDiameter, wirePitch, trenchLength, \
        dotShift, rectangleWidth, trenchDepth, angle, trenchWidth, trenchSeperation, numIdenticalTrenches, \
        stemLength, trenchSeperation = \
            self.OKButtonFunc(wx.EVT_BUTTON) # press OKButton to retrieve data

        Globals.linefullwarning = 0
        self.x = 0 + Globals.xBlockNr * Globals.block_distance
        self.y+= trenchLength+Globals.y_seperation
        self.lineNumber += 1
        self.lineCounter.SetLabel('Current line: %s' % self.lineNumber)

    def NewBlockbuttonFunc(self,event):
        Globals.xBlockNr += 1
        self.x = Globals.xBlockNr*Globals.block_distance
        self.y = 0
        self.blocksCounter.SetLabel('Current block: %s' % int(Globals.xBlockNr + 1))
        self.lineNumber = 1
        self.lineCounter.SetLabel('Current line: %s' % self.lineNumber)
        Globals.blockfullwarning = 0
        Globals.linefullwarning = 0

    def EstimatexSizeFunc(self,event):
```

```
        hashtagSize, aParam, bParam, cParam, s1, s2, wireLength, wireDiameter, wirePitch, trenchLength, \
        dotShift, rectangleWidth, trenchDepth, angle, trenchWidth, trenchSeperation, numIdenticalTrenches, \
        stemLength, trenchSeperation = \
            self.OKButtonFunc(wx.EVT_BUTTON) # press OKButton to retrieve data

        xSizeEstimate = (wireDiameter[1]-wireDiameter[0])/wireDiameter[2]*\
                        (trenchSeperation+(cParam + s1 * hashtagSize[0] + s2 * hashtagSize[1] + trenchWidth)
                         + trenchSeperation)

        numxSizeEstimate = int(Globals.block_size/xSizeEstimate)
        messagexSize = 'The estimated amount of trenches in one line: %s' % numxSizeEstimate
        wx.MessageBox(messagexSize + ' This number might not be fully accurate.', 'Estimate', wx.OK
                      | wx.ICON_INFORMATION)

    def SaveButtonFunc(self,event):
        #If save location is not defined ask for location
        if not self.SaveLocation:
            self.saveLocationFunc()

        # If still no location is defined cancel saving
        if not self.SaveLocation:
            return()

        try:
            waitDlg = wx.BusyInfo('Saving...')
            self.drawing.saveas(self.SaveLocation)
            del waitDlg

        except:
            del waitDlg
            wx.MessageBox('Something went wrong. Maybe the file is still open in autoCAD?','Save error',wx.OK
                          | wx.ICON_INFORMATION)

        self.RealtxtSave = self.SaveLocation[:-3]
        self.RealtxtSave += 'txt'
        if os.path.exists(self.RealtxtSave):
            try:
                os.remove(self.RealtxtSave)
            except:
                msgPath1 = 'A txt file with the same name is open in another program. Close the other program and ' \
                           'restart this one. If you do not, no marker information can be written.'
                dlgPath1 = wx.MessageDialog(parent=None, message=msgPath1, caption='Error')

        dataLogFileReal = open(self.RealtxtSave,'w')

        dataLogFileReal.write(Globals.dataLogFile)
        dataLogFileReal.close()

    def ActualSave(self):
        self.drawing.saveas(self.SaveLocation)

    def defaultValues(self):
        config = configparser.ConfigParser()
        config.read('config.ini')

        self.dlghashtag1size.SetValue('%d' % Functions.ConfigExistChecker('defaults','hashtag1size',config,2))
```

```
        self.dlghashtag2size.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'hashtag2size', config, 2))
        self.dlgaparam.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'a_param', config, 800))
        self.dlgbparam.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'b_param', config, 800))
        self.dlgstemlength.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'stem_length', config, 500))
        self.dlgwirelength.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'wire_length', config, 6000))
        self.dlgwirediametermin.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'wire_min_diameter', config,
            ↪ 20))
        self.dlgwirediametermax.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'wire_max_diameter', config,
            ↪ 50))
        self.dlgwirediameterstep.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'wire_step_diameter', config,
            ↪ 10))
        self.dlgwirepitch.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'wire_pitch', config, 2000))
        self.dlgtrenchlength.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'trench_length', config, 200))
        self.dlgdotshiftmin.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'dotshift_min', config, 0))
        self.dlgdotshiftmax.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'dotshift_max', config, 150))
        self.dlgrectanglewidth.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'rectangle_width', config, 200))
        self.dlgtrenchdepth.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'trench_depth', config, 580))
        self.dlgtrenchseparation.SetValue('%d' % Functions.ConfigExistChecker('defaults', 'trench_separation', config,
            ↪ 5))

        with open('config.ini', 'w') as configfile:
            config.write(configfile)

if __name__ == '__main__':
    app = wx.App()
    EBLWriter(None, title='EBL DXF writer')
    app.MainLoop()
```

Listing B.2: Functions.py

```
import math
import matplotlib.pyplot as plt
from dxfwrite import DXFEngine as dxf
import wx
import Globals
import configparser

def plot_point(point, angle, length):
    x, y = point

    endy = y + length * math.sin(angle)
    endx = x + length * math.cos(angle)


    lines=plt.plot([x, endx], [y, endy])
    plt.setp(lines,'color','r')
    plt.axis('scaled')

#2D plot function to see wire arrangement
def plot_2D(hashtagSize,cParam,s1,s2,angle,wireLength):
# if plt.fignum_exists(1):
# wx.MessageBox('Close the previous plot first.', 'Close previous plot', wx.OK
# | wx.ICON_INFORMATION)
# return()
```

```
    plotfig = plt.figure(1)
    plotfig.canvas.set_window_title('2D NW visualization')
    plotfig.clear()
    plt.xlabel('[nm]')
    plt.ylabel('[nm]')
    for i in range(0, hashtagSize[0]):
        plot_point([-cParam / 2 - s1 * i, 0], angle, wireLength)
    for i in range(0,hashtagSize[1]):
        plot_point([cParam / 2 + s2 * i, 0], math.radians(180.0) - angle, wireLength)

    plt.show()



def DrawSingleTrench(x,y,drawing,hashtagSize,aParam,bParam,cParam,s1,s2,wireLength,wireDiameter,wirePitch,
                     trenchLength,dotShift,rectangleWidth,angle,trenchWidth,stemLength,trenchSeperation,
                         ↪ writeButtonClicks):

    # Calculate useful stuff
    dotsPerTrench = int(trenchLength / wirePitch)
    dotShiftDelta = (dotShift[1]-dotShift[0])/(dotsPerTrench)

    # draw rectangles including compensation for the wire not growing out of rectangle center.
    for i in range(0, hashtagSize[0]):
        drawing.add(dxf.rectangle((x -cParam / 2 - s1 * i
                                   + (trenchWidth - rectangleWidth) * (0.25-0.25*Globals.dots_centering), y),
                                  rectangleWidth, trenchLength,layer='Rectangles'))

    for i in range(0, hashtagSize[1]):
        drawing.add(dxf.rectangle((x +cParam / 2 + s2 * i - rectangleWidth*1
                                   - (trenchWidth - rectangleWidth) * (0.25-0.25*Globals.dots_centering), y),
                                  rectangleWidth, trenchLength,layer='Rectangles'))

    # draw realistic trench size calculated from depth
    for i in range(0, hashtagSize[0]):
        drawing.add(dxf.rectangle((x -cParam/2-s1*i+(trenchWidth - rectangleWidth) * (0.25-0.25*Globals.
            ↪ dots_centering)
                                   -(trenchWidth-rectangleWidth)/2,y),trenchWidth,
                                  trenchLength,layer='TrenchSize'))

    for i in range(0, hashtagSize[1]):
        drawing.add(dxf.rectangle((x +cParam/2+s2*i-rectangleWidth*1
                                   - (trenchWidth - rectangleWidth) * (0.25-0.25*Globals.dots_centering)
                                   -(trenchWidth-rectangleWidth)/2,y),trenchWidth,
                                  trenchLength,layer='TrenchSize'))

    #draw dots. The added sine function is to compensate for the EBL inaccuracy, amplitude determines max offset
    for i in range(0, hashtagSize[0]):
        for j in range(0, dotsPerTrench + 1):
            drawing.add(dxf.rectangle((x-cParam / 2 -wireDiameter/2 - s1 * i+
                                       Globals.sin_amplitude*math.sin(Globals.sin_periods*2*j*math.pi/(
                                           ↪ dotsPerTrench+1)),
                                       y -wireDiameter/2+ j * wirePitch + dotShift[0] + dotShiftDelta*j),wireDiameter,
                                      wireDiameter,layer='Dots'))
```

```
    for i in range(0, hashtagSize[1]):
        for j in range(0, dotsPerTrench + 1):
            drawing.add(dxf.rectangle((x+cParam / 2 -wireDiameter/2 + s2 * i+
                                      Globals.sin_amplitude*math.sin(Globals.sin_periods*2*j*math.pi/(
                                          ↪ dotsPerTrench+1)),
                                      y -wireDiameter/2+ j * wirePitch),wireDiameter,
                                     wireDiameter,layer='Dots'))


    #Shadowing wires
    if Globals.shadowing == 1:
        for i in range(0,hashtagSize[0]):
            for j in range(0, dotsPerTrench + 1):
                drawing.add(dxf.rectangle(
                    (x - cParam / 2 - wireDiameter / 2 - s1 * i +
                     Globals.sin_amplitude * math.sin(Globals.sin_periods*2 * j * math.pi / (dotsPerTrench + 1))-
                     Globals.shadowing_xoffset,
                     y - wireDiameter / 2 + j * wirePitch + dotShift[0] + dotShiftDelta * j+Globals.shadowing_yoffset),
                    wireDiameter*Globals.shadowing_diameter_factor,
                    wireDiameter*Globals.shadowing_diameter_factor, layer='Dots'))
    elif Globals.shadowing == 2:
        for i in range(0,hashtagSize[0]):
            for j in range(0, dotsPerTrench + 1):
                drawing.add(dxf.rectangle(
                    (x - cParam / 2 - wireDiameter / 2 - s1 * i +
                     Globals.sin_amplitude * math.sin(Globals.sin_periods*2 * j * math.pi / (dotsPerTrench + 1)) -
                     Globals.shadowing_xoffset,
                    y - wireDiameter / 2 + j * wirePitch + dotShift[0] + dotShiftDelta * j + Globals.shadowing_yoffset),
                    wireDiameter*Globals.shadowing_diameter_factor,
                    wireDiameter*Globals.shadowing_diameter_factor, layer='Dots'))

                drawing.add(dxf.rectangle(
                    (x - cParam / 2 - wireDiameter / 2 - s1 * i +
                     Globals.sin_amplitude * math.sin(Globals.sin_periods*2 * j * math.pi / (dotsPerTrench + 1))-
                     Globals.shadowing_xoffset,
                     y - wireDiameter / 2 + j * wirePitch + dotShift[0] + dotShiftDelta * j-Globals.shadowing_yoffset),
                    wireDiameter*Globals.shadowing_diameter_factor,
                     wireDiameter*Globals.shadowing_diameter_factor, layer='Dots'))

    #Calculate total size, 1.2 factor for spacing
    xSize = (cParam + s1*hashtagSize[0] + s2*hashtagSize[1] + trenchWidth) + trenchSeperation


    return drawing,xSize

def DrawMultipleDiameterTrench(xBlock,yBlock,drawing,hashtagSize,aParam,bParam,cParam,s1,s2,wireLength,
     ↪ wireDiameter,
                               wirePitch,trenchLength,dotShift,rectangleWidth,angle,trenchWidth,stemLength,
                               trenchSeperation,writeButtonClicks):

    waitDlg = wx.BusyInfo('Drawing...')

    wireDiameterDelta = wireDiameter[1] - wireDiameter[0]
    numDiameterValues = int(wireDiameterDelta/wireDiameter[2])

    x = xBlock
```

```
    y = yBlock

    #Add identifiers
    if writeButtonClicks < 9:
        markerString = '00'+str(writeButtonClicks+1)
    elif writeButtonClicks < 99:
        markerString = '0'+str(writeButtonClicks+1)
    else:
        markerString = str(writeButtonClicks+1)

    drawing.add(dxf.text(markerString,(xBlock-Globals.marker_xoffset,y-Globals.marker_yoffset),
                        height = Globals.marker_height, layer='TextMarkers'))

    # write trenches
    for i in range(0,numDiameterValues+1):
        drawing,xSize = DrawSingleTrench(x,y,drawing,hashtagSize,aParam,bParam,cParam,s1,s2,wireLength,
                                        wireDiameter[0]+wireDiameter[2]*i,
                                    wirePitch,trenchLength,dotShift,rectangleWidth,angle,trenchWidth,stemLength,
                                            trenchSeperation,writeButtonClicks)
        x += xSize



    #Write log file
    Globals.dataLogFile+=markerString+"\n"
    Globals.dataLogFile+="Hashtag size "+ str(hashtagSize[0]) +"x" + str(hashtagSize[1])+"\n"
    Globals.dataLogFile+="a parameter "+str(aParam)+"\n"
    Globals.dataLogFile+="b parameter " + str(bParam) + "\n"
    Globals.dataLogFile+="wirelength " + str(wireLength) + "\n"
    Globals.dataLogFile+="wire diameter " + str(wireDiameter[0])+"; "+str(wireDiameter[1])+"; "+str(wireDiameter[2]) + "
        ↪ \n"
    Globals.dataLogFile+="pitch "+str(wirePitch)+ "\n"
    Globals.dataLogFile+="trench length " + str(trenchLength)+ "\n"
    Globals.dataLogFile+="dot shift " + str(dotShift[0])+"; "+str(dotShift[1])+ "\n"
    Globals.dataLogFile+="rectangle width " + str(rectangleWidth)+ "\n"
    Globals.dataLogFile+="stem length " + str(stemLength)+ "\n"
    Globals.dataLogFile+="trench seperation " + str(trenchSeperation)+ "\n"

    del waitDlg
    return x-xBlock #Return size of the total block

def Configparser():

    config = configparser.ConfigParser()
    config.read('config.ini')

    Globals.block_size = ConfigExistChecker('block_params','block_size',config,500000)
    Globals.block_distance = ConfigExistChecker('block_params','block_distance',config,540000)
    Globals.y_seperation = ConfigExistChecker('trench_params','y_seperation',config,20000)
    Globals.marker_height = ConfigExistChecker('marker_params','marker_height',config,2000)
    Globals.marker_xoffset = ConfigExistChecker('marker_params','marker_xoffset',config,5000)
    Globals.marker_yoffset = ConfigExistChecker('marker_params','marker_yoffset',config,7000)
    Globals.sin_amplitude = ConfigExistChecker('trench_params','sin_amplitude',config,50)
    Globals.sin_periods = ConfigExistChecker('trench_params','sin_periods',config,5)
    Globals.shadowing_xoffset = ConfigExistChecker('trench_params','shadowing_xoffset',config,180)
    Globals.shadowing_yoffset = ConfigExistChecker('trench_params','shadowing_yoffset',config,400)
```

```
    Globals.shadowing_diameter_factor = ConfigExistChecker('trench_params','shadowing_diameter_factor',config,1.5)
        ↪
    Globals.dots_centering = ConfigExistChecker('trench_params','dots_centering',config,0.2)

    with open('config.ini', 'w') as configfile:
        config.write(configfile)

#Check whether config section and option exist, if not create and attach default value
def ConfigExistChecker(section,option,config,defaultvalue):

    while True:
        try:
            globalparam = float(config.get(section,option))
            return globalparam
            break
        except configparser.NoSectionError:
            config.add_section(section)
        except configparser.NoOptionError:
            config.set(section, option, '%g' % defaultvalue)
```

Listing B.3: Globals.py

```
dataLogFile = ""
xBlockNr = 0
shadowing = 0
blockfullwarning = 0
linefullwarning = 0

# ini constants
block_size = 0
block_distance = 0
y_seperation = 0
marker_height = 0
marker_xoffset = 0
marker_yoffset = 0
sin_amplitude = 0
sin_periods = 0
shadowing_xoffset = 0
shadowing_yoffset = 0
shadowing_diameter_factor = 0
dots_centering = 0
```