ChipWhisperer Setup Test

The following notebook can be used to quickly perform a test of your ChipWhisperer capture setup.

If you have downloaded the ChipWhisperer virtual machine, the only real test is if you can connect to the hardware. The VM includes all required tools such as Python modules, compilers, etc. If you are installing "Bare metal", you will need to ensure the compiler and similar are working to follow along with the tutorials.

Jupyter Setup

Presumably you've already got Jupyter running! So that's pretty good start. You can test that a few imports are working by running the following, you shouldn't get any exceptions:

In [1]:
```
import chipwhisperer
help(chipwhisperer)
```

```
Help on package chipwhisperer:

NAME
    chipwhisperer

DESCRIPTION
    .. module:: chipwhisperer
       :platform: Unix, Windows
       :synopsis: Test

    .. moduleauthor:: NewAE Technology Inc.

    Main module for ChipWhisperer.

PACKAGE CONTENTS
    analyzer (package)
    capture (package)
    common (package)
    hardware (package)
    logging

SUBMODULES
    key_text_patterns
    ktp
    programmers
    project
    scopes
    targets
    util

CLASSES
    builtins.object
        StreamPlot

    class StreamPlot(builtins.object)
     |  Methods defined here:
     |
     |  __init__(self)
     |      Initialize self.  See help(type(self)) for accurate signature.
     |
     |  plot(self)
     |
     |  update(self, data)
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

FUNCTIONS
    capture_trace(scope: Union[chipwhisperer.capture.scopes.OpenADC.OpenADC, chipw
hisperer.capture.scopes.cwnano.CWNano], target: Union[chipwhisperer.capture.target
```

```
s.CW305.CW305, chipwhisperer.capture.targets.CW305_ECC.CW305_ECC, chipwhisperer.ca
pture.targets.CW310.CW310, chipwhisperer.capture.targets.SimpleSerial.SimpleSeria
l, chipwhisperer.capture.targets.SimpleSerial2.SimpleSerial2, chipwhisperer.captur
e.targets.SimpleSerial2.SimpleSerial2_CDC], plaintext: chipwhisperer.common.utils.
util.bytearray, key: Optional[chipwhisperer.common.utils.util.bytearray] = None, a
ck: bool = True, poll_done: bool = False, as_int: bool = False, always_send_key=Fa
lse) -> Optional[chipwhisperer.common.traces.Trace]
        Capture a trace, sending plaintext and key

        Does all individual steps needed to capture a trace (arming the scope
        sending the key/plaintext, getting the trace data back, etc.). Uses
        target.output_len as the length of the expected target reponse for
        simpleserial.

        Args:
            scope (ScopeTemplate): Scope object to use for capture.
            target (TargetTemplate): Target object to read/write text from.
            plaintext (bytearray): Plaintext to send to the target. Should be
                unencoded bytearray (will be converted to SimpleSerial when it's
                sent). If None, don't send plaintext.
            key (bytearray, optional): Key to send to target. Should be unencoded
                bytearray. If None, don't send key. Defaults to None.
            ack (bool, optional): Check for ack when reading response from target.
                Defaults to True.
            poll_done (bool, optional): poll Husky to find out when it's done
                capturing, instead of calculating the capture time based on the
                capture parameters. Useful for long trigger-based segmented
                captures.  Can also result in slightly faster captures when the
                number of samples is high. Defaults to False. Supported by Husky
                only.
            as_int (bool, optional): If False, return trace as a float. Otherwise,
                return as an int.
            always_send_key (bool, optional): If True, always send key. Otherwise,
                only send if the key is different from the last one sent.

        Returns:
            :class:`Trace <chipwhisperer.common.traces.Trace>` or None if capture
            timed out.

        Raises:
            Warning or OSError: Error during capture.

        Example:
            Capturing a trace::

                import chipwhisperer as cw
                scope = cw.scope()
                scope.default_setup()
                target = cw.target()
                ktp = cw.ktp.Basic()
                key, pt = ktp.new_pair()
                trace = cw.capture_trace(scope, target, pt, key)

        .. versionadded:: 5.1
            Added to simplify trace capture.
```

```
            .. versionchanged:: 5.2
                Added ack parameter and use of target.output_len

            .. versionchanged:: 5.6.1
                Added poll_done parameter for Husky

        check_for_updates() -> str
            Check if current ChipWhisperer version is the latest.

            Checks pypi.

            .. versionadded:: 5.6.1

        create_project(filename: str, overwrite: bool = False)
            Create a new project with the path <filename>.

            If <overwrite> is False, raise an OSError if this path already exists.

            Args:
               filename (str): File path to create project file at. Must end with .cwp
               overwrite (bool, optional): Whether or not to overwrite an existing
                   project with <filename>. Raises an OSError if path already exists
                   and this is false. Defaults to false.

            Returns:
               A chipwhisperer project object.

            Raises:
               OSError: filename exists and overwrite is False.

        import_project(filename: str, file_type: str = 'zip', overwrite: bool = False)
            Import and open a project.

            Will import the **filename** by extracting to the current working
            directory.

            Currently support file types:
             * zip

            Args:
                filename (str): The file name to import.
                file_type (str): The type of file that is being imported.
                    Default is zip.
                overwrite (bool): Whether or not to overwrite the project given as
                    the **import_as** project.

            .. versionadded:: 5.1
                Add **import_project** function.

        list_devices(idProduct: Optional[List[int]] = None, get_sn=True, get_hw_loc=Tr
    ue) -> List[dict]
            Get a list of devices by NewAE (VID 0x2b3e) currently connected

            Args:
                idProduct (Optional[List[int]], optional): List of PIDs to restrict de
    vices to. If None, do
```

```
                not restrict. Defaults to None.
            get_sn (bool, optional): Whether or not to try to access serial numbe
r. Can fail if another
                process is connected or if we don't have permission to access the
device. Defaults to True.
            get_hw_loc (bool, optional): Whether or not to access the hardware loc
ation of the device.
                Can fail due to the same reasons as above. Defaults to True.

        Returns:
            List[dict]: A list of dicts with fields {'name': str, 'sn', str, 'hw_l
oc': (int, int)}

        If an unknown NewAE device is connected, 'name' will be 'unknown'. If 'sn'
or 'hw_loc'
        are not desired, or cannot be accessed, they will be None.

        .. versionadded:: 5.6.2

    open_project(filename: str)
        Load an existing project from disk.

        Args:
           filename (str): Path to project file.

        Returns:
           A chipwhisperer project object.

        Raises:
           OSError: filename does not exist.

    plot(*args, **kwargs)
        Get a plotting object for use in Jupyter.

        Uses a Holoviews/Bokeh plot with a width of 800 and
        a height of 600. You must have Holoviews and Bokeh
        installed, as well as be working in a Jupyter
        environment.

        args and kwargs are the same as a typical Holoviews plot.

        Plotting a trace in a Jupyter environment::

            import chipwhisperer as cw
            scope = cw.scope()
            ...
            trace = cw.capture_trace(scope, target, text, key)
            display(cw.plot(trace.wave))

        Returns:
            A holoviews Curve object

        .. versionadded:: 5.4

    program_sam_firmware(serial_port: Optional[str] = None, hardware_type: Optiona
l[str] = None, fw_path: Optional[str] = None)
```

```
        Program firmware onto an erased chipwhisperer scope or target

        See https://chipwhisperer.readthedocs.io/en/latest/firmware.html for more
information

        .. versionadded:: 5.6.1
            Improved programming interface

    program_target(scope: Union[chipwhisperer.capture.scopes.OpenADC.OpenADC, chip
whisperer.capture.scopes.cwnano.CWNano], prog_type, fw_path: str, **kwargs)
        Program the target using the programmer <type>

        Programmers can be found in the programmers submodule

        Args:
           scope (ScopeTemplate): Connected scope object to use for programming
           prog_type (Programmer): Programmer to use. See chipwhisperer.programmer
s
               for available programmers
           fw_path (str): Path to hex file to program

        .. versionadded:: 5.0.1
            Simplified programming target

    scope(scope_type: Optional[Type[Union[chipwhisperer.capture.scopes.OpenADC.Ope
nADC, chipwhisperer.capture.scopes.cwnano.CWNano]]] = None, name: Optional[str] =
None, sn: Optional[str] = None, idProduct: Optional[int] = None, bitstream: Option
al[str] = None, force: bool = False, prog_speed: int = 10000000, **kwargs) -> Unio
n[chipwhisperer.capture.scopes.OpenADC.OpenADC, chipwhisperer.capture.scopes.cwnan
o.CWNano]
        Create a scope object and connect to it.

        This function allows any type of scope to be created. By default, the
        object created is based on the attached hardware (OpenADC for
        CWLite/CW1200, CWNano for CWNano).

        Scope Types:
         * :class:`scopes.OpenADC` (Pro and Lite)
         * :class:`scopes.CWNano` (Nano)

        If multiple chipwhisperers are connected, the serial number of the one you
        want to connect to can be specified by passing sn=<SERIAL_NUMBER>

        Args:
            scope_type: Scope type to connect to. Types
                can be found in chipwhisperer.scopes. If None, will try to detect
                the type of ChipWhisperer connected. Defaults to None.
            name: model name of the ChipWhisperer that you want to
                connect to. Alternative to specifying the serial number when
                multiple ChipWhisperers, all of different type, are connected.
                Defaults to None. Valid values:

                * Lite

                * Pro
```

```
                * Husky

            idProduct: idProduct of the ChipWhisperer that you want to
                connect to. Alternative to specifying the serial number when
                multiple ChipWhisperers, all of different type, are connected.
                Defaults to None. Valid values:

                * 0xace2: CW-Lite

                * 0xace3: CW-Pro

                * 0xace5: CW-Husky

            sn: Serial number of ChipWhisperer that you want to
                connect to. sn is required if more than one ChipWhisperer of the
                same type is connected (i.e. two CWNano's or a CWLite and CWPro).
                Defaults to None.
            bitstream: Path to bitstream to program. If None,
                programs default bitstream. Optional, defaults to None. Ignored
                on Nano.
            force: If True, always erase and program
                FPGA. If False, only erase and program FPGA if it
                is currently blank. Defaults to False. Ignored on Nano.
            prog_speed: Sets the FPGA programming speed for Lite, Pro, and Husky.
                If you get programming errors, try turning this down.

        Returns:
            Connected scope object.

        Raises:
            OSError: Can be raised for issues connecting to the chipwhisperer, suc
h
                as not having permission to access the USB device or no ChipWhispe
rer
                being connected.
            Warning: Raised if multiple chipwhisperers are connected, but the type
                and/or the serial numbers are not specified

        .. versionchanged:: 5.1
            Added autodetection of scope_type

        .. versionchanged:: 5.5
                Added idProduct, name, bitstream, and force parameters.

    target(scope: Union[chipwhisperer.capture.scopes.OpenADC.OpenADC, chipwhispere
r.capture.scopes.cwnano.CWNano, NoneType], target_type: type = <class 'chipwhisper
er.capture.targets.SimpleSerial.SimpleSerial'>, **kwargs) -> Union[chipwhisperer.c
apture.targets.CW305.CW305, chipwhisperer.capture.targets.CW305_ECC.CW305_ECC, chi
pwhisperer.capture.targets.CW310.CW310, chipwhisperer.capture.targets.SimpleSeria
l.SimpleSerial, chipwhisperer.capture.targets.SimpleSerial2.SimpleSerial2, chipwhi
sperer.capture.targets.SimpleSerial2.SimpleSerial2_CDC]
        Create a target object and connect to it.

        Args:
           scope (ScopeTemplate): Scope object that we're connecting to the target
               through.
```

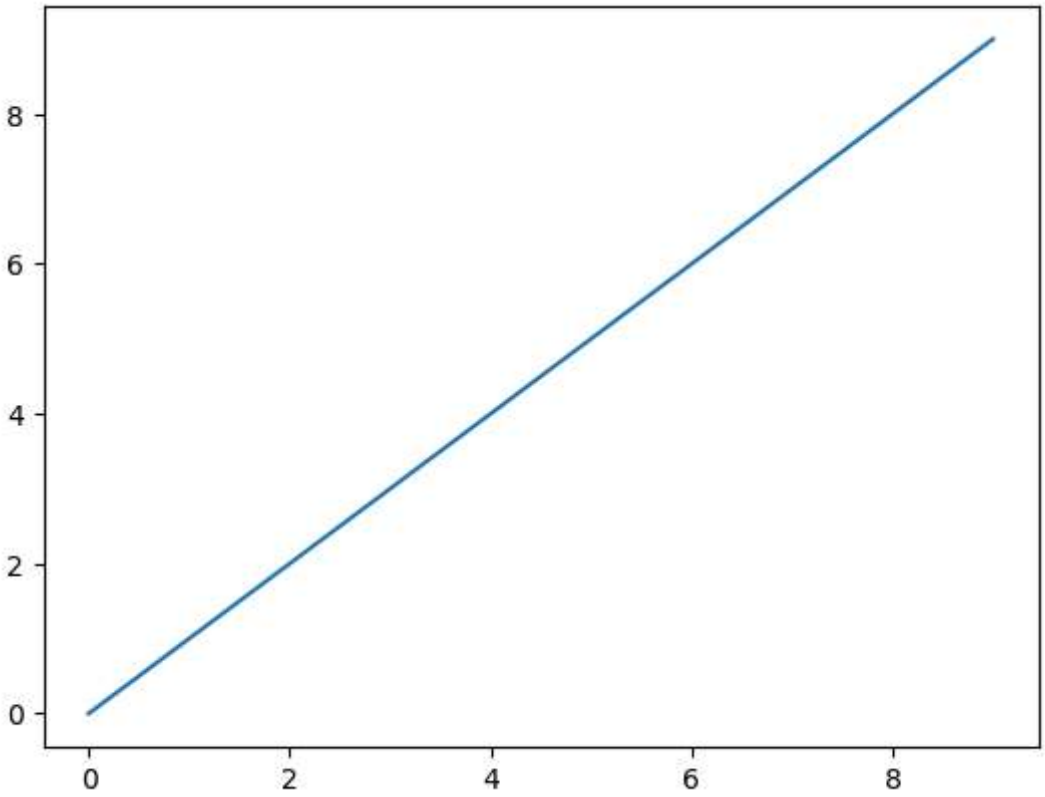
```
            target_type (TargetTemplate, optional): Target type to connect to.
                Defaults to targets.SimpleSerial. Types can be found in
                chipwhisperer.targets.
            **kwargs: Additional keyword arguments to pass to target setup. Rarely
                needed.

        Returns:
            Connected target object specified by target_type.

DATA
    List = typing.List
        A generic version of list.

    Optional = typing.Optional
        Optional type.

        Optional[X] is equivalent to Union[X, None].

    Type = typing.Type
        A special construct usable to annotate class objects.

        For example, suppose we have the following classes::

          class User: ...  # Abstract base for User classes
          class BasicUser(User): ...
          class ProUser(User): ...
          class TeamUser(User): ...

        And a function that takes a class argument that's a subclass of
        User and returns an instance of the corresponding class::

          U = TypeVar('U', bound=User)
          def new_user(user_class: Type[U]) -> U:
              user = user_class()
              # (Here we could write the user object to a database)
              return user

          joe = new_user(BasicUser)

        At this point the type checker knows that joe has type BasicUser.

    Union = typing.Union
        Union type; Union[X, Y] means either X or Y.

        To define a union, use e.g. Union[int, str].  Details:
        - The arguments must be types and there must be at least one.
        - None as an argument is a special case and is replaced by
          type(None).
        - Unions of unions are flattened, e.g.::

            Union[Union[int, str], float] == Union[int, str, float]

        - Unions of a single argument vanish, e.g.::

            Union[int] == int  # The constructor actually returns int
```

```
        - Redundant arguments are skipped, e.g.::

            Union[int, str, int] == Union[int, str]

        - When comparing unions, the argument order is ignored, e.g.::

            Union[int, str] == Union[str, int]

        - You cannot subclass or instantiate a union.
        - You can use Optional[X] as a shorthand for Union[X, None].

    chipwhisperer_loggers = [<Logger ChipWhisperer NAEUSB (WARNING)>, <Log...
    cw_formatter = <logging.Formatter object>
    filehndlr = <FileHandler C:\Users\j8023\AppData\Local\Temp\c...whisper...
    glitch_logger = <Logger ChipWhisperer Glitch (WARNING)>
    log_dir = r'C:\Users\j8023\AppData\Local\Temp\chipwhispererjw1htrkk'
    logger = <Logger ChipWhisperer Glitch (WARNING)>
    naeusb_logger = <Logger ChipWhisperer NAEUSB (WARNING)>
    other_logger = <Logger ChipWhisperer Other (WARNING)>
    scope_logger = <Logger ChipWhisperer Scope (WARNING)>
    strmhndlr = <StreamHandler stderr (DEBUG)>
    target_logger = <Logger ChipWhisperer Target (WARNING)>
    tracewhisperer_logger = <Logger ChipWhisperer TraceWhisperer (WARNING)...

VERSION
    5.6.1

FILE
    c:\users\j8023\chipwhisperer5_64\cw\home\portable\chipwhisperer\software\chipw
hisperer\__init__.py
```

The following should generate a plot - **NOTE: You may need to run this multiple times for some reason**

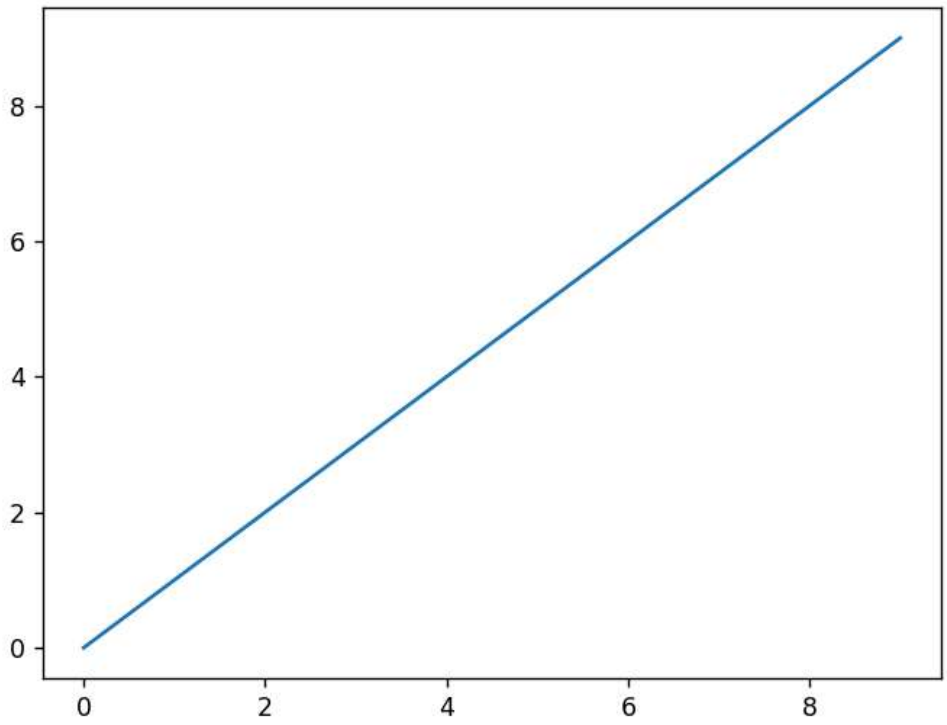
In [2]:
```
import matplotlib.pylab as plt
plt.plot(range(0, 10))
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x240dd711090>]
```



The following should generate an interactive plot

In [3]:
```
%matplotlib notebook
import matplotlib.pylab as plt
plt.plot(range(0, 10))
```



Out[3]: `[<matplotlib.lines.Line2D at 0x240dd8169b0>]`

ChipWhisperer Hardware Test

The following will connect to a ChipWhisperer. Just plug the Chipwhisperer main board in itself (black PCB or ChipWhisperer-Pro box), this isn't testing any of the attached target boards or similar. If the ChipWhisperer communication is up, everything else should "just work" when talking to various targets.

This should result in some text that says `INFO: Found ChipWhisperer😍`

In [4]:
```
PLATFORM="CWNANO"
%run Setup_Scripts/Setup_Generic.ipynb
```

```
INFO: Found ChipWhisperer😍
```

We aren't going to use the hardware, so should disconnect from it so it is dropped by this Python kernel.

In [5]:
```
scope.dis()
target.dis()
```

Bash / Command-Line Test

The following will check you have a working bash or command-line interface. We'll be using this for running `make` and other such commands.

In [6]:
```
%%sh
dir
```

```
0\ -\ Introduction\ to\ Jupyter\ Notebooks.ipynb  Test_Notebook.py
1\ -\ Connecting\ to\ Hardware.ipynb              archive
ChipWhisperer\ Firmware\ Upgrade.ipynb           asdf.txt
ChipWhisperer\ Setup\ Test.ipynb                 courses
ChipWhisperer\ Updating.ipynb                    demos
Helper_Scripts                                   img
README.md                                        requirements.txt
SYSC5807\ Assignment\ on\ Module\ #4             tests
Setup_Scripts                                    user
Test_Notebook.ipynb
```

In [7]:
```
!dir
```

```
 Volume in drive C is OS
 Volume Serial Number is 0698-35F2

 Directory of C:\Users\j8023\ChipWhisperer5_64\cw\home\portable\chipwhisperer\jupy
ter

2024-04-06  12:51 PM    <DIR>          .
2024-04-06  12:42 PM    <DIR>          ..
2024-04-06  12:46 PM    <DIR>          .git
2023-01-16  05:38 PM               928 .gitignore
2024-03-27  01:13 PM    <DIR>          .ipynb_checkpoints
2024-04-06  12:51 PM           205,315 0 - Introduction to Jupyter Notebooks.ipynb
2023-01-16  05:38 PM            20,821 1 - Connecting to Hardware.ipynb
2024-04-06  12:43 PM    <DIR>          archive
2023-01-16  05:38 PM                 0 asdf.txt
2023-01-16  05:38 PM             1,500 ChipWhisperer Firmware Upgrade.ipynb
2023-01-16  05:38 PM             6,963 ChipWhisperer Setup Test.ipynb
2023-01-16  05:38 PM             5,237 ChipWhisperer Updating.ipynb
2024-04-06  12:43 PM    <DIR>          courses
2024-04-06  12:43 PM    <DIR>          demos
2024-04-06  12:43 PM    <DIR>          Helper_Scripts
2024-04-06  12:43 PM    <DIR>          img
2023-01-16  05:38 PM             3,204 README.md
2023-01-16  05:38 PM               215 requirements.txt
2024-04-06  12:43 PM    <DIR>          Setup_Scripts
2024-03-27  05:21 PM    <DIR>          SYSC5807 Assignment on Module #4
2024-04-06  12:43 PM    <DIR>          tests
2023-01-16  05:38 PM             8,504 Test_Notebook.ipynb
2023-01-16  05:38 PM             6,714 Test_Notebook.py
2024-04-06  12:43 PM    <DIR>          user
              11 File(s)        259,401 bytes
              13 Dir(s)  182,665,388,032 bytes free
```

Remember that Jupyter allows running commands on your system - so exposing the web interface can be dangerous! This is one reason ChipWhisperer-Install on Windows uses a Token by default, and the VM forces you to set a password. Unless you have

port forwarding your firewall should prevent access remotely, but if using this on a hostile network be sure you have not opened the port!

Compiler Testing

If you'll be building source code, you need a working Make system. First check you can run `make` - this should give you the normal output of

In [8]:
```
!make --version
```

```
GNU Make 4.2.1
Built for Windows32
Copyright (C) 1988-2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

If you are using any of the following, you will need the `arm-none-eabi-gcc` compiler:

- ChipWhisperer-Lite 32-bit (with Arm Target).
- ChipWhisperer UFO Board with any Arm target (such as STM32F3, STM32F0, etc).
- ChipWhisperer-Nano

You can easily check for the working compiler with the following, which sohuld print the version and build information:

In [9]:
```
!arm-none-eabi-gcc -v
```

```
Using built-in specs.
COLLECT_GCC=arm-none-eabi-gcc
COLLECT_LTO_WRAPPER=c:/users/j8023/chipwh~1/cw/home/portable/armgcc/bin/../lib/gc
c/arm-none-eabi/10.2.1/lto-wrapper.exe
Target: arm-none-eabi
Configured with: /mnt/workspace/workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-
48_20201124_1606180641/src/gcc/configure --build=x86_64-linux-gnu --host=i686-w64-
mingw32 --target=arm-none-eabi --prefix=/mnt/workspace/workspace/GCC-10-pipeline/j
enkins-GCC-10-pipeline-48_20201124_1606180641/install-mingw --libexecdir=/mnt/work
space/workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180641/ins
tall-mingw/lib --infodir=/mnt/workspace/workspace/GCC-10-pipeline/jenkins-GCC-10-p
ipeline-48_20201124_1606180641/install-mingw/share/doc/gcc-arm-none-eabi/info --ma
ndir=/mnt/workspace/workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_
1606180641/install-mingw/share/doc/gcc-arm-none-eabi/man --htmldir=/mnt/workspace/
workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180641/install-m
ingw/share/doc/gcc-arm-none-eabi/html --pdfdir=/mnt/workspace/workspace/GCC-10-pip
eline/jenkins-GCC-10-pipeline-48_20201124_1606180641/install-mingw/share/doc/gcc-a
rm-none-eabi/pdf --enable-languages=c,c++ --enable-mingw-wildcard --disable-decima
l-float --disable-libffi --disable-libgomp --disable-libmudflap --disable-libquadm
ath --disable-libssp --disable-libstdcxx-pch --disable-nls --disable-shared --disa
ble-threads --disable-tls --with-gnu-as --with-gnu-ld --with-headers=yes --with-ne
wlib --with-python-dir=share/gcc-arm-none-eabi --with-sysroot=/mnt/workspace/works
pace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180641/install-mingw/
arm-none-eabi --with-libiconv-prefix=/mnt/workspace/workspace/GCC-10-pipeline/jenk
ins-GCC-10-pipeline-48_20201124_1606180641/build-mingw/host-libs/usr --with-gmp=/m
nt/workspace/workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1606180
641/build-mingw/host-libs/usr --with-mpfr=/mnt/workspace/workspace/GCC-10-pipelin
e/jenkins-GCC-10-pipeline-48_20201124_1606180641/build-mingw/host-libs/usr --with-
mpc=/mnt/workspace/workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_20201124_1
606180641/build-mingw/host-libs/usr --with-isl=/mnt/workspace/workspace/GCC-10-pip
eline/jenkins-GCC-10-pipeline-48_20201124_1606180641/build-mingw/host-libs/usr --w
ith-libelf=/mnt/workspace/workspace/GCC-10-pipeline/jenkins-GCC-10-pipeline-48_202
01124_1606180641/build-mingw/host-libs/usr --with-host-libstdcxx='-static-libgcc -
Wl,-Bstatic,-lstdc++,-Bdynamic -lm' --with-pkgversion='GNU Arm Embedded Toolchain
10-2020-q4-major' --with-multilib-list=rmprofile,aprofile
Thread model: single
Supported LTO compression algorithms: zlib
gcc version 10.2.1 20201103 (release) (GNU Arm Embedded Toolchain 10-2020-q4-majo
r)
```

If you are using any of the following, you will need the `avr-gcc` compiler:

- ChipWhisperer-Lite Classic (with XMEGA Target).
- ChipWhisperer UFO Board with XMEGA or AVR Target.
- ChipWhisperer-Lite 2-Part Version.

You can easily check for the working compiler with the following, which should print the version and build information:

In [10]:
```
!avr-gcc -v
```

```
Using built-in specs.
Reading specs from c:/users/j8023/chipwh~1/cw/home/portable/avrgcc/bin/../lib/gcc/
avr/11.1.0/device-specs/specs-avr2
COLLECT_GCC=avr-gcc
COLLECT_LTO_WRAPPER=c:/users/j8023/chipwh~1/cw/home/portable/avrgcc/bin/../libexe
c/gcc/avr/11.1.0/lto-wrapper.exe
Target: avr
Configured with: ../configure --prefix=/omgwtfbbq/avr-gcc-11.1.0-x64-windows --tar
get=avr --enable-languages=c,c++ --disable-nls --disable-libssp --disable-libada -
-with-dwarf2 --disable-shared --enable-static --enable-mingw-wildcard --enable-plu
gin --with-gnu-as --host=x86_64-w64-mingw32 --build=x86_64-pc-linux-gnu
Thread model: single
Supported LTO compression algorithms: zlib
gcc version 11.1.0 (GCC)
```

Conclusions

That's it! We've got a working ChipWhisperer system now. There are a few packages used in specific tutorials that could still be missing if you've done your own install, but the above should have validate all important system settings. Hopefully any of the "important to setup" stuff should have been shaken out already.