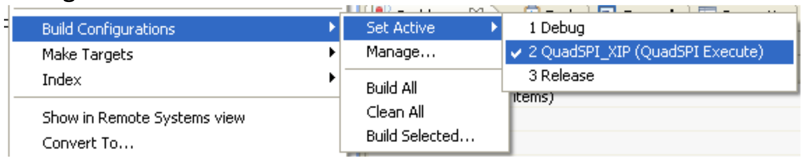
The **quadspi_loader** project will take a binary and flash it into the QuadSPI on the TWR-VF65GS10. For DS-5, a c-array is used instead of a binary.

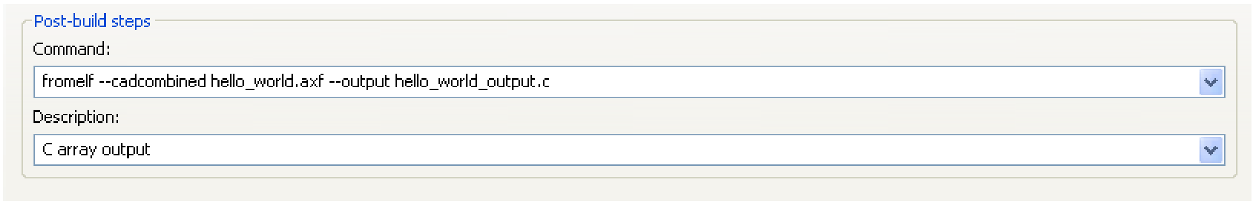
DS-5 Details

The QuadSPI_XIP build configuration for the hello_world target adds the special Image Vector Table (.ivt) and Boot Data sections to the code, which are required when booting from QuadSPI.

Select the QuadSPI_XIP configuration by right clicking on the hello_world project and selecting Build Configuration->Set Active.



A c-array is output when this target is built by using the “fromelf” image converter. Right click project, go to Properties->Build Steps



Post-build steps

Command:

```
fromelf --cadcombined hello_world.axf --output hello_world_output.c
```

Description:

C array output

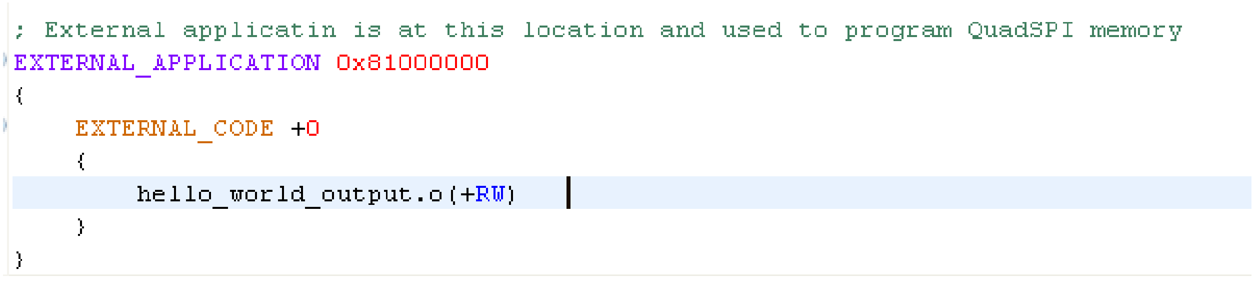
Using that target adds the special Image Vector Table (IVT) and Boot Data required to boot from QuadSPI to your code.

This file is compiled into the quadspi_load target. The section in the scatter file into a specific section, at different locations for SRAM and DDR:

SRAM:

```
; External applicatin is at this location and used to program QuadSPI memory
EXTERNAL_APPLICATION 0x3F400000
{
    EXTERNAL_CODE +0
    {
        hello_world_output.o(+RW)
    }
}
```

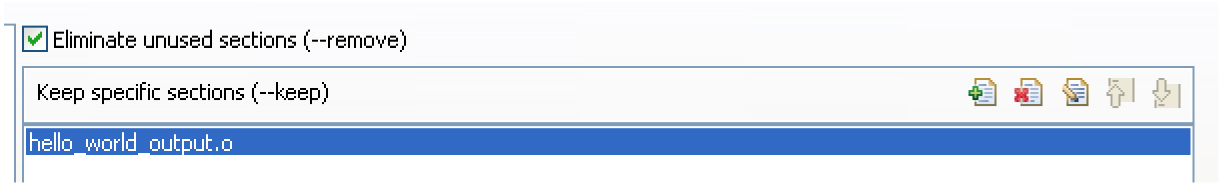
DDR:



```
; External applicatin is at this location and used to program QuadSPI memory
EXTERNAL_APPLICATION 0x81000000
{
    EXTERNAL_CODE +0
    {
        hello_world_output.o(+RW)
    }
}
```

The locations for both targets (0x3F400000, 0x81000000) are simply placeholders for the image which was compiled to the QuadSPI location 0x20000000.

To guarantee these sections aren't removed at compile-time, we need to tell DS-5 to keep the array. Right click project, select Properties->Settings->ARM Linker->Optimizations:



✓ Eliminate unused sections (--remove)

Keep specific sections (--keep)

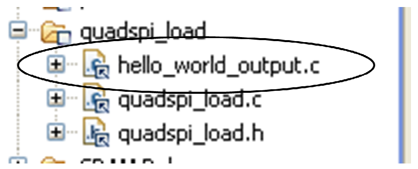
hello_world_output.o

Use the DDR target if your QuadSPI image is over 1Mb.

The location of the QuadSPI base address is imported into the source code using the following method (quadspi_load.c)

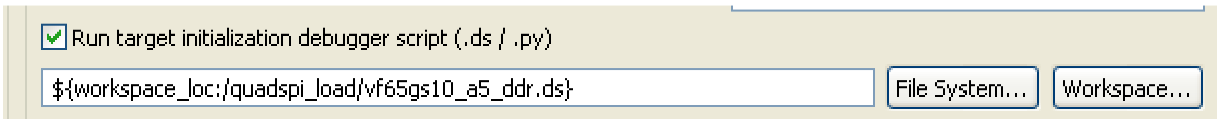
```
extern unsigned int Image$$EXTERNAL_CODE$$Base[];  /* Use Linker Symbol */
#define PROGRAM_SOURCE_LOCATION            (uint32)Image$$EXTERNAL_CODE$$Base
```

The hello_world_ouptut c-array is located in the src folder but was imported as a linked resource.



```
quadspi_load
    hello_world_output.c
    quadspi_load.c
    quadspi_load.h
```

Lastly, under Debug Configurations, the quadspi_load DDR target runs an additional initialization script to setup the DRAM controller



✓ Run target initialization debugger script (.ds / .py)

${workspace_loc:/quadspi_load/vf65gs10_a5_ddr.ds}