# Example SOC – Build Instructions

## OVERVIEW

These instructions detail how to build and simulate the Example SOC, which will be the basis of your top-level testing in the main coursework. You will be given the RTL for some additional peripherals that you will need to modify according to the specification that will be provided and then integrate these into the Example SOC framework.

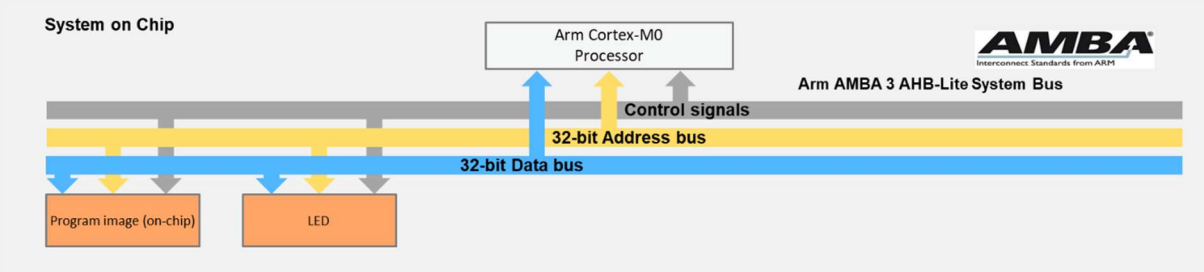
- Hardware implementation:
  - Implement the hardware framework that has been written in Verilog.
- Software programming:
  - Program the Cortex-M0 processor using assembly language.
  - Simulate the program using QuestaSim.
- Demonstrate the SoC:
  - Toggle the 8-bit LEDs at a given frequency.

## DETAILS

### HARDWARE

The hardware components of the SoC include:

- An Arm Cortex-M0 microprocessor
- An AHB-Lite system bus
- Two AHB peripherals
  - Program memory (implemented using on-chip memory blocks)
  - A simple LED peripheral




**SoC Peripherals**

- Arm Cortex-M0 microprocessor:

The logic of the Arm Cortex-M0 processor is written in Verilog code, and thus can be simulated in QuestaSim. In this example for teaching purposes, we use a simplified version of the Cortex-M0 processor, called Cortex-M0 DesignStart. The Cortex-M0 DesignStart has almost the same functionality of an industry-standard Cortex-M0 processor, except that

some features are reduced; e.g., the number of interrupts is reduced from the original 32 to 16 interrupts.

- On-chip program memory:

To program a processor, your software code needs to be compiled to machine code, which are the instructions to be executed by the processor. The physical memory used to store these instructions is called a program memory. In this basic SoC platform, the program memory is implemented using the on-chip memory blocks, rather than an off-chip memory. For example, the block RAM (BRAM) is one type of on-chip memory.

Normally, in order to load your program into the on-chip memory of the example SOC, the program image needs to be merged into your hardware design during synthesizing. For example, if you need to preload a program file into the hardware, the program file (e.g., “code.hex”) needs to be referred to in your Verilog code, using syntax such as:

```
initial begin
$readmemh("code.hex", memory);
end
```

We will use this same technique, although you will only be simulating the design rather than synthesizing to implement it in an FPGA or ASIC.

- LED peripheral:

The LED peripheral is a simple module used to interface with the 8-bit LEDs. It has an AHB bus interface, which allows the LED to be connected to the system AHB bus, and controlled by the Cortex-M0 processor.

The files needed in this lab are listed below:

FILES TO BE USED IN THIS LAB

| **Components** | **File name** | **Description** |
|---|---|---|
| Cortex-M0 processor | cortexm0ds_logic.v | Cortex-M0 DesignStart processor logic level Verilog file: The DesignStart is a simplified version used for education. |
| | CORTEXM0INTEGRATION.v | Cortex-M0 DesignStart processor macro cell level |

| | | |
|---|---|---|
| AHB bus component | AHBDCD.v | The address decoder of the AHB bus |
| | AHBMUX.v | The slave multiplexor of the AHB bus |
| AHB on-chip memory peripheral | AHB2BRAM.v | The on-chip memory (BRAM) used for the program memory of the processor |
| AHB LED peripheral | AHB2LED.v | The LED peripheral module |
| Top module | AHBLITE_SYS.v | The top-level module |

You will need to download the Cortex-M0 DesignStart package from the Arm website.

Go to this webpage:

https://www.arm.com/resources/designstart/designstart-university

Then under "Arm CPU Evaluation", click 'Learn More'

Then under Cortex-M0, click on 'Apply Now'

You'll then need to create an account before you can download the **Cortex-M0 DesignStart Eval Package** (you need to agree to the license terms)

That should be a file called AT510-MN-80001-r2p0-00rel0

From that just the two files for the Cortex-M0 processor Verilog are needed that you'll find in the cores/cortexm0_designstart_r2p0/logical/cortexm0_integration/verilog/ directory.

The other Verilog files you will find on in the Files tab on Teams, under 'Example SOC'

The memory map can be defined in the AHB bus address decoder. Below is the default memory map of the two peripherals:

MEMORY MAP OF PERIPHERALS

| Peripheral | Base address | End address | Size |
|---|---|---|---|
| MEM | 0x0000_0000 | 0x0000_FFFF | 16MB |
| LED | 0x5000_0000 | 0x50FF_FFFF | 16MB |

## SOFTWARE

In this example, we will use the assembly language to program the Cortex-M0 processor. Assembly language allows us to access the registers at a low level; hence, we can have a better understanding of the low-level hardware mechanism.

| File name | Description |
|---|---|

| | |
|---|---|
| cm0dsasm.s | The assembly code used in this lab |

The main code should perform the following:

- Initialize the interrupt vector.
- In the reset handler, repeat the following:
  - Turn on half of the 8-bit LEDs, e.g., LED [0, 2, 4, 6].
  - Set a counter, and use it to delay for a short time.
  - Turn on the other half of the LEDs, e.g., LED [1, 3, 5, 7].
  - Delay for another period.

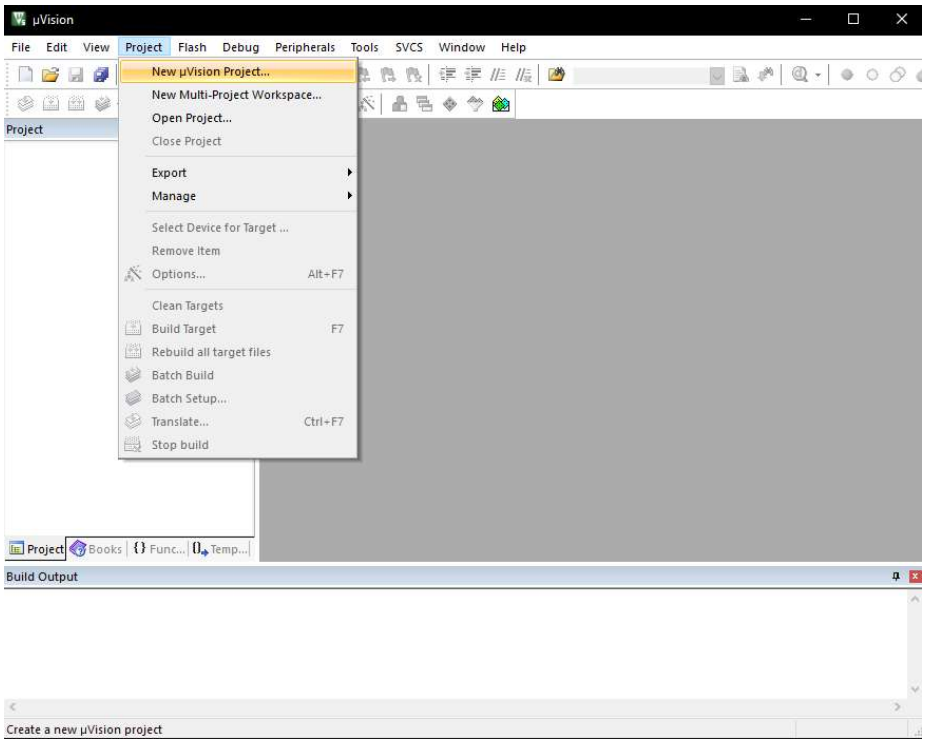
## TESTING

You can use an evaluation copy of the Keil tools to assemble your program.

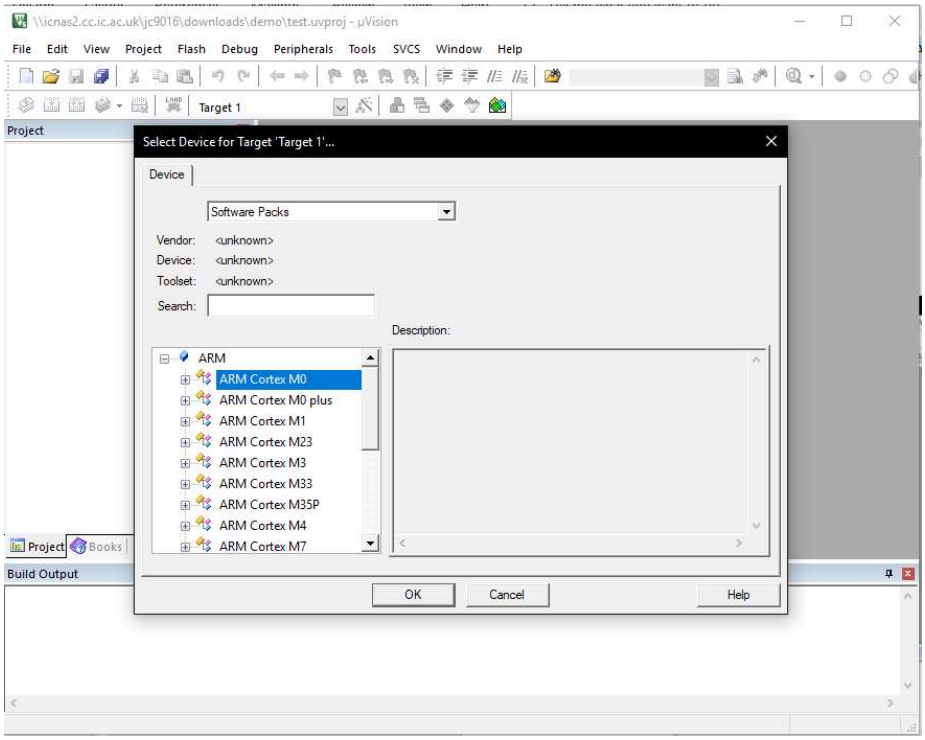
Download the Keil tools from http://www2.keil.com/mdk5/selector

Click 'Download' under 'Community'

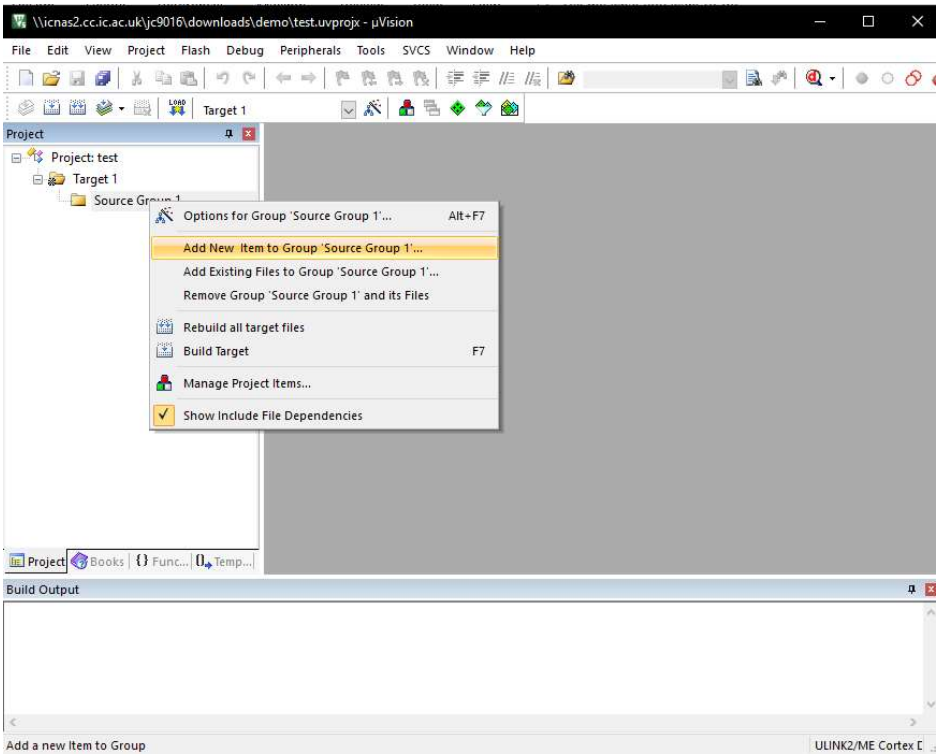
Open the Keil tool and create a project, click Project > New µVision Project:

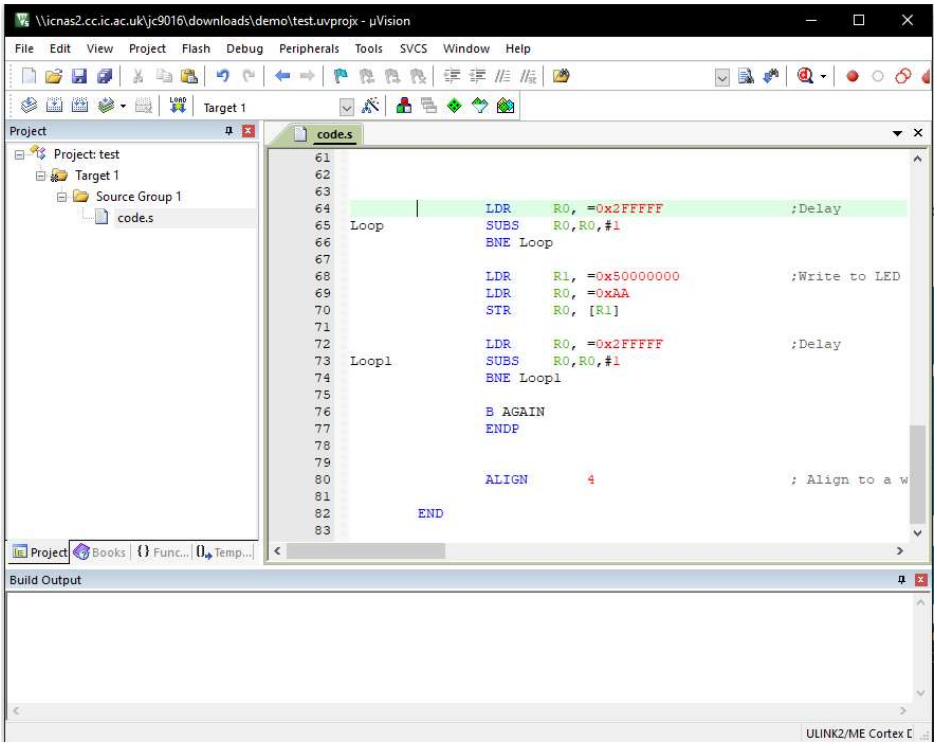



Select **ARM Cortex M0** as your target and click **OK**:

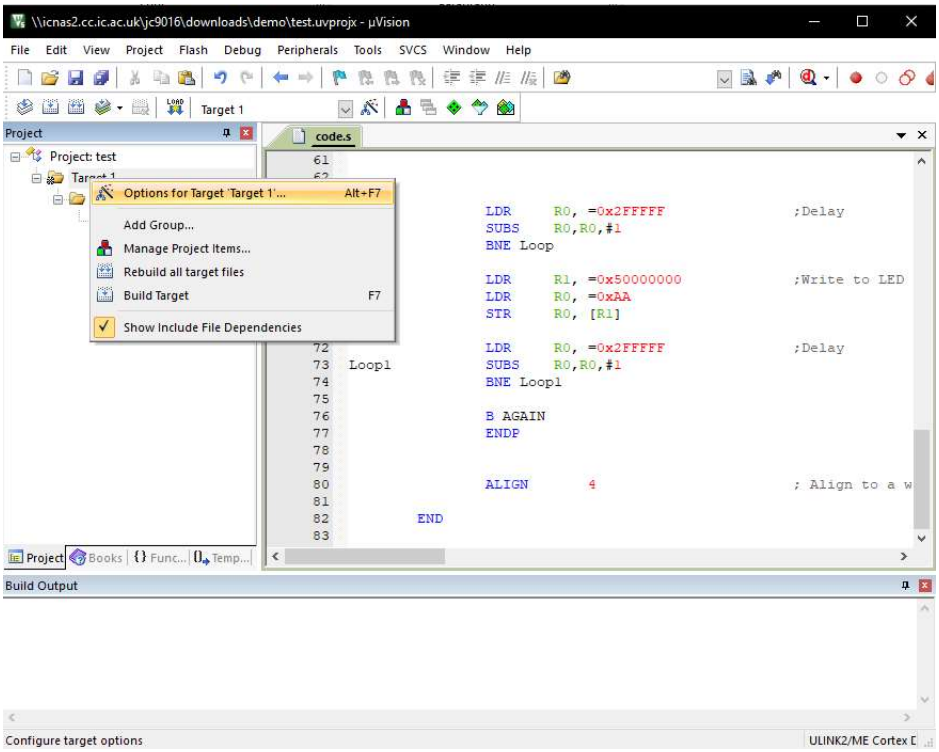



Now a project is created. Right click to create a new file and start to write your own code (you can use the supplied code for the LED peripheral):



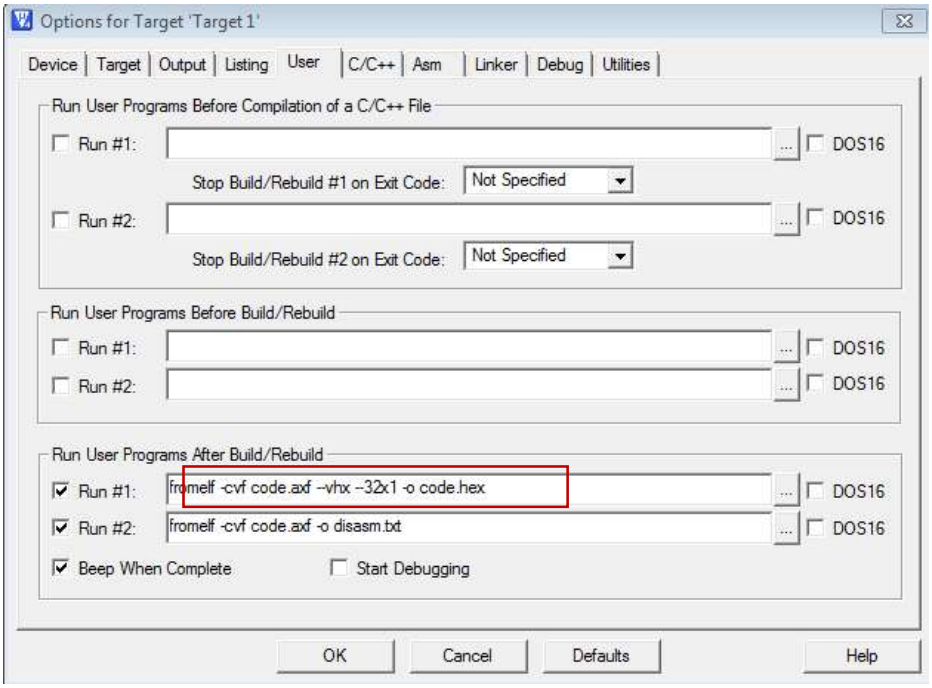



Now we want to compile the code into a hex code. Firstly we need to configure the command after building target. **Right click** the target and select **Options for Target "Target 1":**

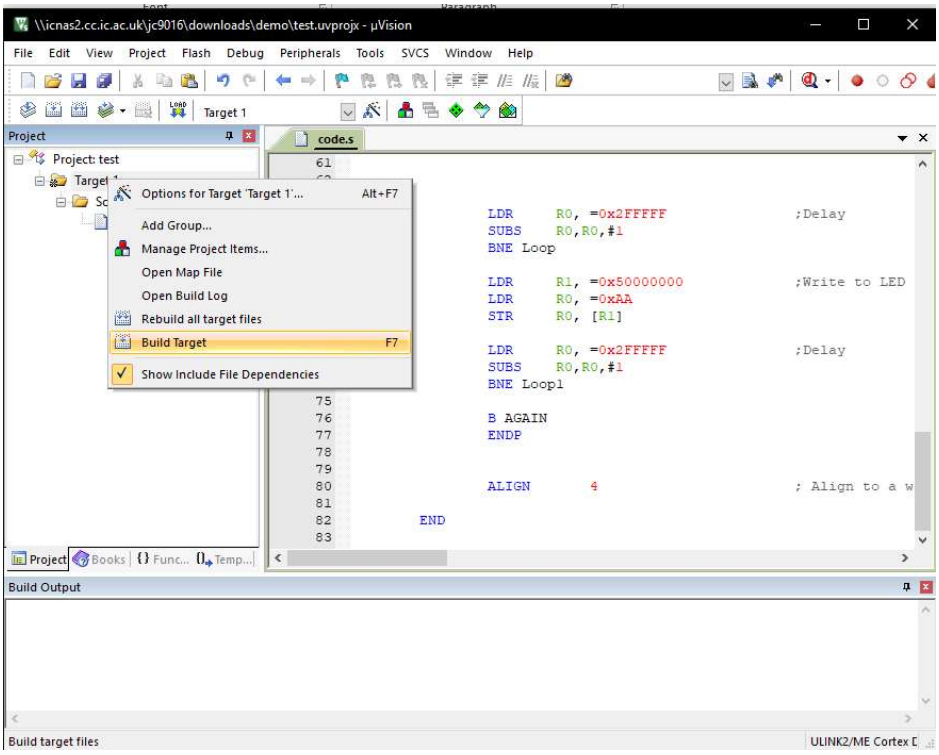



In the User tab, **check Run #1**. Add the following command and click **OK**:

fromelf -cvf code.axf --vhx --32x1 -o code.hex




Now we build the code. Right click the **Target > Build Target**:




Copy the executable file “code.hex” to the root project directory (i.e. the directory where you are running the simulation).

Run the simulation in Questasim, as for the first coursework. In this case use ‘vlog -f ahblite_sys.vc’ when compiling the code.

After reset, you should see the LED bus in the simulation toggling.

# HARDWARE DEBUGGING

## HARDWARE LOGIC SIMULATION

We can use hardware simulation tools to analyze the system behavior, in this case using Mentor QuestaSim.

The simulation tool allows you to analyze a set of signals. The suggested signals are:

- HADDR[31:0]
- HWDATA[31:0]
- HRDATA[31:0]
- HWRITE
- HREADY
- HSIZE[2:0]
- HTRANS[1:0]
- HRESP

# EXTENSION WORK

Here are some extra things that you can do (but are not part of this coursework):

- Add additional registers to the LED peripheral. For example, add a mask register that can mask out certain bits when writing the LEDs.
- Add another peripheral “AHB switch” to input the status of the 8-bit switches. For example, use the switch to control the LEDs.