Programming of XMOS devices on FlashCore3

Version: 1.0, 10.2015

The programming solution of DATA I/O for XMOS devices with OTP memory is based on the SPI boot capability of these devices. The SPI flash image is used to boot the xCORE, program the non-volatile memory and then signal the result of the programming process via GPIO ports of the device. This is the XMOS recommended process for production programmer.

This application note will guide you how to setup the SPI flash image using the XMOS xTIMEcomposer ™ tools. This image will be programmed into the SPI flash memory on the adapter.

XMOS tools required for creating the SPI flash image:

- xTIMEcomposer ™ (https://www.xmos.com/products/tools/xtimecomposer)

Following command line tools are part of the xTIMEcomposer development environment:

- **XCC:** it is the front-end to the xCORE C, C++ and XC compilers.
- **XOBJDUMP:** it is used for inspection and updating of the contents of XE files.
- **XBURN:** it creates OTP images and programs images into the OTP memory
- **XFLASH:** it creates binary files in the xCORE flash format.

This document is based on the XMOS application note AN00153 - Programming OTP memory via SPI boot.

Typical programming flow for OTP memory via SPI flash:

1. Generate xCORE application executable for target (tool: xcc)
2. Generate raw OTP binary image for xCORE application (tool: xobjdump, xburn)
3. Build OTP programming executable for programming adapter (tool: xburn)
4. Build SPI flash image for programming OTP memory (tool: xflash)
5. Program external SPI slave with OTP programming image
6. Power on the xCORE device and boot from SPI flash image
7. Check programming signals for completion and PASS/FAIL

Steps 1 and 2 should be straight forward for the XMOS application developer.

Step 3: Once the binary image for the OTP was created the OTP programming image can be generated, which will perform the actual writing and verification of data into the OTP memory. This is done by passing the following code to xburn on the command line:

```
xburn --make-exec otp_programmer.xe --extra-file prog_interface.xc
otp_image.bin --target-file dataio.xn
```

prog_interface.xc defines the signaling method for the result of the programming process.

dataio.xn describes DATA I/O's target platform in XML.

For the source code of prog_interface.xc and dataio.xn refer to appendix A please.

Step 4 converts the OTP programming image into an SPI flash boot image:

```
xflash -o otp_programmer.bin otp_programmer.xe --noinq --boot-
partition-size 0x10000 --verbose
```

This command produces a binary image in the correct format and can be loaded via TLWin to be programmed into the SPI slave device (included on the programming adapter).

Steps 5-6-7 are executed by the FlashCore3 programmer, after the programming job with the SPI flash image as data file input was loaded into the programmer by TLWin.

APPENDIX A

prog_interface.xc

```
#include <platform.h>

out port complete_signal = XS1_PORT_8B;

void _done()
{
  set_port_use_on(complete_signal);
  complete_signal <: 0x5A;  //success code
  while (1);
}


void _exit(int exitcode)
{
  set_port_use_on(complete_signal);

  if (exitcode != 0)
  {
    complete_signal <: 0xFF;
  }
  else
  {
    complete_signal <: 0x5A;  //success code
  }
  while (1);
}
```

Notes:

- The standard runtime function _done() is overloaded by this code to report the status that programming the OTP has completed successfully. This function is called by the OTP programming application when there has been no error in programming.
- The standard runtime function _exit(…) is overloaded by this code to report the status that programming the OTP has completed and has either passed or failed. This function is called by the OTP programming application when there has been an error in programming.

Dataio.xn

```
<?xml version="1.0" encoding="UTF-8"?>
<Network xmlns="http://www.dataio.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.dataio.com http://www.dataio.com">
 <Type>Device</Type>
 <Name>XS1-L4A-64-TQ48-C4 Device</Name>
 <Declarations>
  <Declaration>tileref tile[1]</Declaration>
 </Declarations>
 <Packages>
  <Package id="0" Type="XS1-LnA-64-TQ48">
   <Nodes>
    <Node Id="0" InPackageId="0" Type="XS1-L4A-64" SystemFrequency="320MHz" RoutingId="0">
     <Boot>
      <Source Location="SPI:bootFlash"/>
      <Bootee NodeId="1" Tile="0"/>
     </Boot>
     <Tile Number="0" Reference="tile[0]">
      <Port Location="XS1_PORT_1A" Name="PORT_SPI_MISO"/>
      <Port Location="XS1_PORT_1B" Name="PORT_SPI_SS"/>
      <Port Location="XS1_PORT_1C" Name="PORT_SPI_CLK"/>
      <Port Location="XS1_PORT_1D" Name="PORT_SPI_MOSI"/>
     </Tile>
    </Node>
   </Nodes>
  </Package>
 </Packages>
 <ExternalDevices>
  <Device NodeId="0" Tile="0" Name="bootFlash" Class="SPIFlash" Type="W25X40">
   <Attribute Name="PORT_SPI_MISO" Value="PORT_SPI_MISO"/>
   <Attribute Name="PORT_SPI_SS" Value="PORT_SPI_SS"/>
   <Attribute Name="PORT_SPI_CLK" Value="PORT_SPI_CLK"/>
   <Attribute Name="PORT_SPI_MOSI" Value="PORT_SPI_MOSI"/>
  </Device>
 </ExternalDevices>
 <JTAGChain>
  <JTAGDevice NodeId="0"/>
 </JTAGChain>
</Network>
```

APPENDIX B

Lockbuild.bat - batch file example for generating the SPI image for a “secure boot” application:

```
@ECHO OFF
REM XMOS: Build image for SPI flash - LOCK OTP

IF "%1"=="" GOTO ERR_PARAM
IF "%2"=="" GOTO ERR_PARAM

IF NOT EXIST %1.key GOTO not_found_1
IF NOT EXIST %2.xn GOTO not_found_2

REM remove scrap
IF EXIST *.xe DEL *.xe
IF EXIST *.elf DEL *.elf
IF EXIST *.bin DEL *.bin
IF EXIST program_info.txt DEL program_info.txt
IF EXIST xscope.xscope DEL xscope.xscope
IF EXIST config.xml DEL config.xml


xburn -f --jtag-speed 10 --spi-div 25 --target-file %2.xn --lock %1.key -o otp_image.bin

PAUSE

xburn --make-exec otp_programmer.xe --extra-file prog_interface.xc otp_image.bin --target-file
dataio.xn

PAUSE

xflash -o %2_otp_programmer.bin otp_programmer.xe --noinq --boot-partition-size 0x10000 --
verbose

GOTO END

:not_found_1
ECHO File %1.xc not found!
GOTO END

:not_found_2
ECHO File %2.xn not found!
GOTO END

:ERR_PARAM
ECHO Parameter missed!
ECHO Usage: lockbuild key_file target_file
ECHO        All filenames without ending!!!
ECHO The result target_otp_programmer.bin can be loaded in TaskLink.

:END
```