Forth on the ARM Cortex-M1 FPGA Development Kit

Leon H. Wagner
FORTH, Inc.
www.forth.com

Abstract

This paper describes the instantiation of an ARM Cortex-M1 CPU core on an Altera Cyclone III FPGA and the development of a simple Forth application to run on it.

The CPU core used here is the ARM Cortex-M1 FPGA Development Kit from ARM, Ltd. The Altera Quartus II environment is used to design an ARM system, including memory and embedded peripherals. SwiftX-ARM is used to develop and interactively test a simple Forth application on the newly instantiated Cortex-M1 core in the FPGA.

Section 1: Overview

1.1 About SwiftX

SwiftX is FORTH, Inc.'s interactive cross compiler for the development of applications for embedded microprocessors and microcontrollers. SwiftX is based on the Forth programming language and is itself written in Forth.

SwiftX has been ported to the following microprocessor and microcontroller families:

- Atmel, Cirrus, Nuvoton, NXP, ST Microelectronics (and other) ARM cores
- Freescale ColdFire
- Freescale 6801 / Renesas 6303
- Freescale 6809
- Freescale 68HC11
- Freescale 68HC12 (S12, S12X, etc.)
- Freescale 68HCS08
- Freescale 68K
- Aeroflex UTMC 69R000
- Intel (NXP, SiLabs, others) 8051
- Atmel AVR
- Renesas H8H (H8/300H, H8S)
- Intel (AMD, other) i386

- Texas Instruments MSP430
- Patriot PSC1000
- Harris RTX2010

SwiftX cross-compilers run in the SwiftForth programming environment. They inherit all the features of SwiftForth and extend its interactive development environment to manage multiple program and data spaces as well as to generate the code and data that fill them.

1.2 About the ARM Cortex-M1 FPGA Development Kit

The ARM Cortex-M1 FPGA Development Kit, available free of charge from ARM, Ltd., is delivered as an SOPC Builder design optimized for the Altera Cyclone III FPGA Starter Kit. The ARM system bus has been adapted to the Altera Avalon system interface for this implementation. However, there are no architectural changes from the standard ARM Cortex-M1 Core.

The Cortex-M1 processor is intended for deeply embedded applications that require a small processor (i.e., low gate count) integrated into an FPGA. The processor core implements the ARM architecture v6-M Thumb Instruction Set Architecture (ISA) with some 32-bit Thumb-2 extensions.

ARM Cortex-M1 FPGA Development Kit is fully compatible with Altera's SOPC Builder and Quartus II tools. This application note demonstrates how to build a simple system from the ARM Cortex-M1 FPGA Development Kit and a few of the standard Altera peripheral IP blocks, then generate a bitfile of the whole system and program it onto a Cyclone III device.

1.3 Requirements

The following items are required to implement the system described in this application note:

- SwiftX-ARM from FORTH, Inc. (www.forth.com)
- Quartus II (subscription or web edition), version 8.0 or later, from Altera. We used Quartus II Web Edition 11.0 for this paper. (www.altera.com)
- ARM Cortex-M1 FPGA Development Kit. (www.arm.com)
- Cyclone III Starter FPGA Kit. (www.altera.com)
- A simple USB-to-serial port converter, such as the DEV-09873 from Sparkfun Electronics. (www.sparkfun.com)

Section 2: Working with Quartus II and SOPC Builder

This section describes the development of a Quartus II project using the SOPC Builder tool to fabricate the ARM Cortex-M1 core with memory and peripherals. The system is synthesized and downloaded to a Cyclone III FPGA Starter Kit for testing.

We are using the Windows version of Quartus II and are placing our project files in the directory **c:\projects**. Make any necessary adjustments to path names if you are using the Linux version of Quartus II.

2.1 Starting a new Quartus II Project

Launch Quartus II, dismissing any startup wizards or tips. Select File > New Project Wizard and fill in the blanks as follows:

- Set the working directory for the project to **C:\projects\cm1\fpga**.
- Name the new project **cm1**.
- Set the top-level design entity name to **cm1_top**.
- Click the “Next” button (say “Yes” to create the new project directory).
- When you get to the “Add Files” step, just skip it for now. We’ll add some files after we build the ARM Cortex-M1 system in SOPC builder.
- For the “Family & Device Settings” step, select the device that matches the one on your Cyclone III FPGA Starter Kit (e.g., EP3C25F324C6).
- Click “Next” through the remaining wizard screens and “Finish” on the last one.

2.2 Working with SOPC Builder

The Quartus II SOPC Builder tool is used to build and integrate the ARM Cortex-M1 CPU, clocks, memory, PIO, and UART components. The Cortex-M1 CPU core is supplied by ARM, Ltd. The remainder of the IP components are from Altera and are distributed with the Quartus II system.

Select Tools > SOPC Builder in Quartus II and follow the directions in the following sections to build the ARM Cortex-M1 system.

For “System Name” in the “Create New System” dialog box, enter “cm1” and select Verilog as the HDL. Do not use the same name assigned to the top-level entity.

2.2.1 Adding the ARM Cortex-M1 core

Look for “ARM Cortex-M1 Processor” in the Component Library pane of the SOPC Builder window, select it as shown in Figure 1 and click the “Add” button[1].

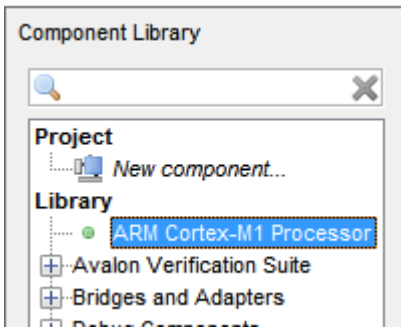

Figure 1. ARM Cortex-M1 Processor component selection

Configure the ARM Cortex-M1 processor as follows:

- Uncheck the “Debug enabled” box
- Set number of IRQs to 8
- Set ITCM size to 16 kB
- Uncheck “Read only” for the ITCM
- Check “Initialize ITCM contents”
- Set ITCM “From file” field to **itcm.hex** (this should be the default)
- Set DTCM size to 8 kB
- Uncheck “Initialize DTCM contents”
- Click the “Finish” button to add the ARM Cortex-M1 core to the SOPC design

2.2.2 Adding parallel I/O (PIO)

The SwiftX demo for the Cortex-M1 will use a 4-bit PIO to drive the four user LEDs and read the four user pushbuttons on the Cyclone III FPGA Starter Kit board.

From the Component Library, select Peripherals > Microcontroller Peripherals > PIO and click the “Add” button. Configure the PIO component as follows:

- Set the width to 4 bits
- Set direction to “InOut”
- Set the output port reset value to 0x0F
- Leave the rest of the PIO option boxes unchecked
- Click the “Finish” button to add the PIO to the SOPC design

2.2.3 Adding a UART

The SwiftX Interactive Development Environment uses a fast serial port as its

1.If the ARM Cortex-M1 Processor item is not present, use Tools > Options to add the path to ARM/CortexM1_DevKit/Component/arm_avalon_cortexm1 to the IP Search Path.

Cross-Target Link (XTL) debug interface.

From the Component Library, select Interface Protocols > Serial > UART (RS-232 Serial Port)[1]. Configure the UART component as follows:

- Under “Basic Settings,”, use the defaults (no parity, 8 bits, 1 stop bit, 2 synchronizer stages, no RTS/CTS, no EOP).
- Select a fixed baud rate of 115200 with error tolerance of 0.01
- Leave the remaining options unchecked
- Set the transmitter baud rate to “Actual” (not “Accelerated”)
- Click the “Finish” button to add the UART to the SOPC design

2.2.4 Connecting the components

In SOPC builder, click on the “Filter” button and set the Filter pull-down to “All” so you can see all the connections.

Set the bus, clocks, and interrupt connections as well as the component base addresses and IRQ as shown in Figure 2.

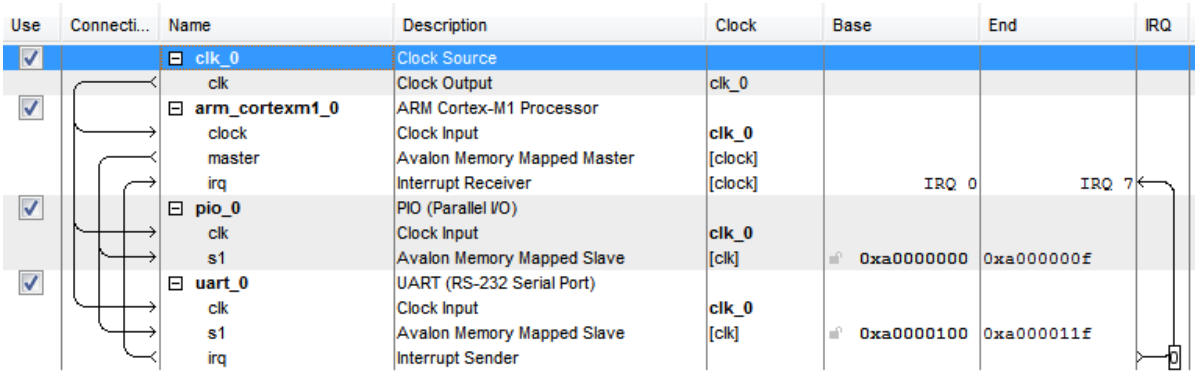

Figure 2. Component connections and settings

The base address for the PIO is 0xA0000000 and the UART is 0xA0000100. Use IRQ 0 for the UART “interrupt sender” element.

2.2.5 Generate the SOPC system

Click the “Generate” button to generate the system. When prompted to save the file, name it **cm1.sopc**.

After the generation is complete, you should see the “System generation was successful” message in SOPC Builder’s output pane. Click the “Exit” button to close the SOPC Builder window, saving any changes to the **cm1.sopc** file.

1.Do not use the JTAG UART component.

2.3 Adding the Top-Level Design

We need to add a top-level design file to make some connections to the ARM Cortex-M1 core and peripherals generated by SOPC Builder. We'll do this with a simple block diagram/schematic file.

Select File > New and choose "Block Diagram/Schematic File" in the New dialog box, then click "Ok." A blank block1.bdf workspace should appear in the right window pane. Follow these steps to create and save the top-level design file:

- Double-click in the center of the schematic grid workspace. This should open the Symbol dialog.
- In the "Libraries" pane, expand the "Project" item and select cm1. This should show the cm1 block entity in the right pane.
- Click "Ok".
- Position the block roughly in the center of the schematic and click to position it.

Now we need to tie a few of the unused inputs to Vcc and Gnd levels:

- Double-click on the schematic to open the Symbol dialog.
- Expand the Quartus Libraries item and select Primitives > Other > Vcc.
- Click on the schematic to place Vcc next to the cm1 node's DBGRESETn input.
- Select a Node tool and connect Vcc to DBGRESETn.
- Double-click on the schematic to open the Symbol dialog.
- Expand the Quartus Libraries item and select Primitives > Other > Gnd.
- Click on the schematic to place Gnd next to the cm1 node's EDBGRQ input.
- Select a Node tool and connect Gnd to both EDBGRQ and NMI inputs.
- Use the Selection tool and select the entire cm1 object.
- Right-click on cm1 and select "Generate Pins for Symbol Ports."
- Save the schematic file as **cm1_top.bdf.**
- In the Project Navigator pane, select the Files tab, right-click on **cm1_top.bdf** and select "Set as Top-Level Entity."

Figure 3 shows a representation of the top-level design schematic.

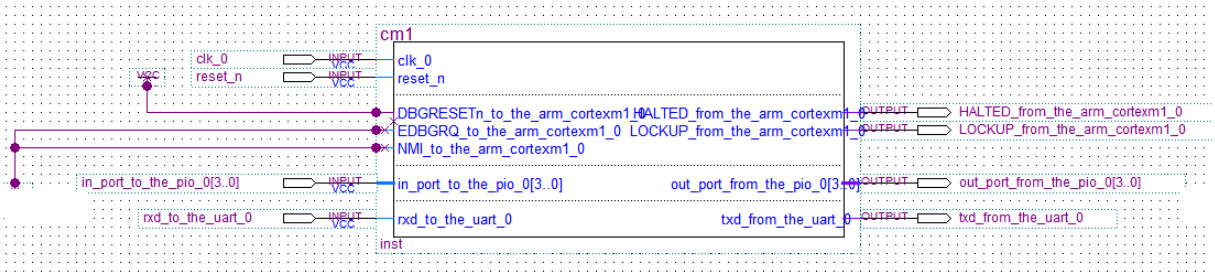

Figure 3. Top-Level design schematic

2.4 Synthesizing the System

The files generated by SOPC Builder along with our top-level design need to be analyzed so we can assign pin connections to the outside world.

Select Processing > Start > Start Analysis and Elaboration so Quartus II can analyze the design and figure out its pins and connections. The process should complete with the message "Analysis & Elaboration was successful."

2.4.1 Assigning nodes and pins

- Select Assignments > Assignment Editor to make the pin assignments.
- In the Assignment Editor window, click on a cell in the "To" column and select Node Finder. Select "Pins: all" ("Look in:" should be **cm1_top**) and click "List" to see all the internal pins.
- Select the pins listed in the "Signal" column of Figure 4 and add them to the pane on the right.
- In the assigment editor, set the Assignment Name field for all pins to "Location" and enter the physical pins listed in Figure 4 into the Value column and set the Enabled column to "Yes" for each pin.
- Close the Assignment Editor, saving the new assignments when prompted.

NET NAME	SIGNAL	PIN
50MHZ	clk_0	V9
CPU_RST_N	reset_n	N2
HSMC_SCL	rxd_to_the_uart_0	F3
HSMC_SDA	txd_from_the_uart_0	E1
KEY0	in_port_to_the_pio_0[0]	F1
KEY1	in_port_to_the_pio_0[1]	F2
KEY2	in_port_to_the_pio_0[2]	A10
KEY3	in_port_to_the_pio_0[3]	B10
LED0	out_port_from_the_pio_0[0]	P13
LED1	out_port_from_the_pio_0[1]	P12
LED2	out_port_from_the_pio_0[2]	N12
LED3	out_port_from_the_pio_0[3]	N9

Figure 4. Pin assignments

2.4.2 Compiling the system

Copy the **itcm.hex** object file from the **projects\cm1\firmware** project directory to the **projects\cm1\fpga** directory. If there is no **itcm.hex file** in the SwiftX project directory, launch the SwiftX project in that directory and do a "Build" to compile the ARM Cortex-M1 SwiftX kernel.

Then go back to the Quartus II window and select Processing > Start Compilation. The compilation step takes a few minutes to complete. When it is done,

there will be some warnings in the output window, but there should be no error messages.

2.4.3 Programming the FPGA

Connect a USB port on the computer running Quartus II to the USB Blaster embedded on the Cyclone III FPGA Starter Kit board. Select Tools > Programmer and under Hardware Setup, select the USB Blaster, setting its mode to JTAG. Select the **cm1.sof** file[1] and click the “Start” button. Figure 5 shows the programmer dialog box after loading the FPGA.

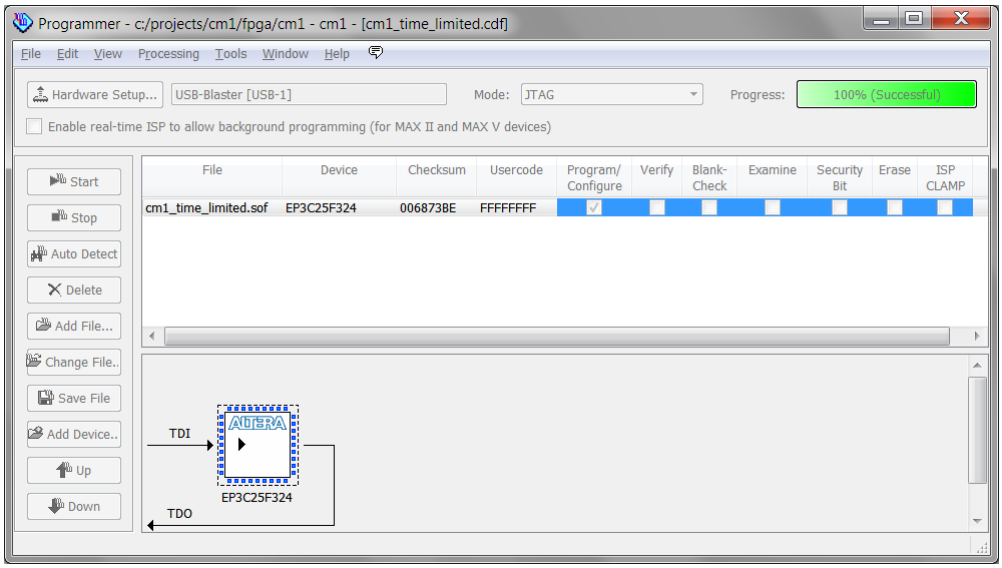

Figure 5. Programmer dialog box

The target board should start running the ARM Cortex-M1 SwiftX program from **itcm.hex**.

1.The exact name of the SOF file may vary.

Section 3: Working with SwiftX

3.1 Connecting the debug interface

A serial port is used for the debug interface from the SwiftX Interactive Development Environment to the target ARM Cortex-M1 CPU inside the Cyclone III FPGA. The four pins on the Cyclone III FPGA Start Kit board originally intended for I2C connections have been reassigned in our design to be used as the UART pins. The pin assignments are as follows:

SDA	TXD
SCL	RXD
3V3	N/C
GND	GND

Connect a USB cable from the computer running SwiftX to a USB-to-serial converter[1] connected to the target board as noted above.

3.2 Establishing an interactive session

Click on the "Debug" button in the SwiftX tool bar (or select Tools > Debug or press the F9 shortcut key) to establish a serial connection with the target CPU and launch an interactive debug session. The output in the SwiftX debug window should look something like this:[2]

```
INCLUDE DEBUG

   Start      End      Size     Used   Unused    Type   Name
    0000     3FFF     16384     7640     8744   CDATA  PROG
20000000 200010FF      4352       12     4340   IDATA  IRAM
20000100 20001FFF      7936     2316     5620   UDATA  URAM

TARGET READY
SwiftX/ARM Cortex-M1 Altera  ok
```

Interactive development and testing can now proceed as described in the *SwiftX Reference Manual* and the *SwiftX ARM Target Reference Manual*.

3.3 Project source files

The project directory **projects\cm1\firmware** contains the usual set of files as documented in the the *SwiftX Reference Manual*. In addition to these, the following project-specific files are supplied:

1.TTL levels required.
2.Exact memory usage may vary.

- **reg_fpga.f** — Defines the memory-mapped register interface to the Altera PIO and UART components.
- **xtl_uart0.f** — Implements the target side of the serial cross-target link (XTL) debug interface using the Altera UART added to the design in 2.2.3.
- **leds.f** — Uses the Altera PIO core component to interface to the four user LEDs on the Cyclone III Starter FPGA Kit board.
- **buttons.f** — Uses the Altera PIO core component to interface to the four user pushbuttons on the Cyclone III Starter FPGA Kit board.
- **demo.f** — Implements a high-level **WINK** function and assigns a "walking" LED wink pattern to a SwiftX **BACKGROUND** task.

The output file generated by the SwiftX **BUILD** process is **itcm.hex**. It must be copied or moved into the **projects\cm1\fpga** directory prior to compiling the FPGA project in Quartus, as described in 2.4.2.

3.4 Demo Application

The demo application consists of a background task that drives the LEDs in a "chase" pattern. The user pushbuttons are tested in the main application loop to change the speed of the chasing pattern (by setting the LED on/off time to a longer or shorter period) and to perform an "all on" LED test.

Excerpts from the Forth source files are included in the next section.

Conclusion

The instantiation of the ARM Cortex-M1 processor on an Altera FPGA using SOPC Builder is a simple task that can be accomplished even by someone with limited FPGA programming experience.

Additional memory and peripherals can be added in SOPC Builder for a much more elaborate implemenetation. Custom logic can be included in the design from the top level using conventional FPGA tools.

Porting a Forth cross compiler and interactive debug environment to this CPU is no more daunting than a port to a conventional CPU.

Section 4: Program Listings

4.1 LED Outputs

```
{ -------------------------------------------------------------------
LED control

!LED writes a pattern to the 4 LEDS in bits [3:0] of the PIO output
register.
------------------------------------------------------------------- }

: !LEDS ( x -- )   INVERT PIO_BASE ! ;
: /LEDS ( -- )   0 !LEDS ;
```

4.2 Button inputs

```
{ -------------------------------------------------------------------
Read switches

?PRESSED returns true if button defined by mask is pressed.
BUTTON1..BUTTON4 define masks for passing to ?PRESSED.
------------------------------------------------------------------- }

: ?PRESSED ( mask -- flag )   PIO_BASE @ AND 0= ;

1 CONSTANT BUTTON1
2 CONSTANT BUTTON2
4 CONSTANT BUTTON3
8 CONSTANT BUTTON4
```

4.3 Demo program

```
{ -------------------------------------------------------------------
LED demo program

LEDTIME holds the number of milliseconds for LED on and off period.

FAST, MEDIUM, and SLOW are the number of milliseconds for three speeds.
SPEED sets LEDTIME to u milliseconds.

CHECK-BUTTONS does the following based on button(s) pressed:
  Button 1: Set slow speed
  Button 2: Set medium speed
  Button 3: Set fast speed
  Button 4: Turn all LEDs on

LED-CHASE performs one loop of the LED "chase" sequence.
CHECK-BUTTONS is called just before the on or off delay time.

CHASER is the task assiged to perform the chasing LEDs behaviour.
/CHASER starts the task.
DEMO initializes the LEDS and time period, then starts the task.
------------------------------------------------------------------- }

VARIABLE LEDTIME        \ Millisconds on/off time

50 CONSTANT FAST
100 CONSTANT MEDIUM
250 CONSTANT SLOW

: SPEED ( u -- )   LEDTIME ! ;
```

```
: CHECK-BUTTONS ( -- )
   BUTTON1 ?PRESSED IF  SLOW SPEED  THEN
   BUTTON2 ?PRESSED IF  MEDIUM SPEED  THEN
   BUTTON3 ?PRESSED IF  FAST SPEED  THEN
   BUTTON4 ?PRESSED IF  $0F !LEDS  THEN ;

: LED-CHASE ( -- )   1
   4 0 DO  DUP !LEDS  CHECK-BUTTONS  LEDTIME @ MS  2*  LOOP
   4 0 DO  2/ DUP !LEDS  CHECK-BUTTONS  LEDTIME @ MS  LOOP DROP ;

|U| |S| |R| BACKGROUND CHASER

: /CHASER ( -- )   CHASER ACTIVATE
   BEGIN  LED-CHASE  AGAIN ;

: DEMO ( -- )
   0 !LEDS  MEDIUM SPEED  CHASER BUILD  /CHASER ;
```