Author : Bertrand CUZEAU – ALSE – http://www.alse-fr.com

A.L.S.E. Application Note

Implementing Multiple Configurations on Altera FPGAs using the Remote Update Circuitry

Upgraded to Intel Cyclone 10 LP

Introduction

This Application Note was created in 2010 to demonstrate how to implement an Altera FPGA-based system that has a Factory-default and a User configuration, thus enabling the possibility of the FPGA to “update itself” reliably.

The idea is to set up and use the “Remote Update IP core” available for most Altera FPGA families.
Unfortunately, this block is not trivial and the available documentation is incomplete and misleading.
This ApNote does not intend to replace the documentation, but it just shows the code & steps for a simple running on Cyclone III using Active Serial configuration.
Many other cases exist, but using this example as a base makes it easier to derive a working strategy for the other FPGAs and other programming schemes.

NB: this ApNote and the archive was created in 2010 using Quartus II version 9.1 sp2, 10.0 and 10.1sp1.
It has been updated in 2011 for the BeMicro SDK kit and to demonstrate the use of the Watchdog function.

In 2023, this ApNote remains useful since the **Cyclone 10 LP family** introduced by Intel after having acquired Altera actually consists of repackaged Cyclone III chips (it was an excellent family). But while the support of Cyclone III in Quartus has been discontinued in 2013, it has been re-introduced for the Cyclone 10 LP family which is supported by the latest Quartus versions.

Note that with our GEDEK IP, **Remote Update through Ethernet** is possible without using any processor !

Reference Documentation

Before complaining about the documentation (or lack thereof), the first step is to search for it... and read it, or at least give it a serious try. The latest Altera now Intel documentation is still insufficient, hence this ApNote.
But it’s interesting to read :

Cyclone 10 LP Device Overview and : **Cyclone 10 LP Handbook** **chapter 6.5** and more important :
Remote Update Intel FPGA IP User Guide in the Cyclone IV and Cyclone 10 LP section.
Other FPGA families will have specific information about the Remote Upgdate feature.

The most useful document is actually the original ApNote from Altera : AN 521: Cyclone III Active Parallel Remote System Upgrade Reference (but make sure you do not use the original Verilog code provided !
It is using a *Parallel* Flash but the control logic of the Remote Update IP is very similar.

If some links are broken, you should be able to retrieve the information using the titles.

Principle

The idea is to store two (or more) different bitstreams in the same Flash device so your FPGA can “boot” using either configuration upon some conditions, and typically:

1. a Factory default (“Safe”) configuration image.
   This design *must* include the Remote Update logic with permission to write into it, this enabling to load another image. It *can* also include the Flash update logic to reprogram the User image remotely.
2. a “User” configuration image, loaded by the Safe image, on some conditions.
   This design may not even include the Remote Update logic.

In a first use case, the Factory default usually permits to remotely reprogram the User image (bitstream), and acts as a fail-safe system in case the reprogramming fails. An example is updating the hardware configuration through Ethernet, like when using ALSE's Gigabit Ethernet IP with Serial or Parallel Flash Programming, or through a Serial link, or though PCIe (other IPs available at ALSE).

A second use case is to have a system starting up and performing some functions, then “rebooting” to another selected configuration image for example based on a DIP switch or on a real-time User commands.

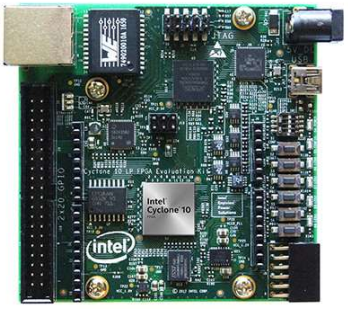
Both use cases involve the same use of the Remote Update Megafunction, so we'll take the second for our example. Note that in active parallel and active serial (AS) modes, you cannot load the User configuration first and fall back to Factory if it goes wrong: *you always need to load Factory first*, and this design would automatically try and load the User configuration, and fall back to Factory (Safe) if this fails.

Among all the different programming schemes, one of the most popular and probably less intuitive to deal with is the **Active Serial** programming using Advanced Serial Configuration (serial NOR Flash) devices like the EPCS. That's what we'll use here.

And to let anyone try the concept on a board, we selected initially one of the cheapest Altera FPGA boards available in 2011 : the Terasic DE0 Board.
As of 2023, this old board is still available but you’ll have to use an old version of Quartus (2013).

We also tested on a BeMicroSDK kit (Cyclone IV), still supported by Quartus

In 2023, we have upgraded this ApNote to the Intel **Cyclone 10 LP Evaluation Kit**. As of Jan 2023, this kit is unfortunately not available for purchase.



The principle of this example on DE0 is that :

- The kit powers up with a design named “**Conf1**”, located as usual at address 0 of the Configuration Flash. This design includes the reconfiguration logic.
- It displays “1” and a blinking decimal point on the seven segments LEDs.
- It also reads from the push buttons.
- When Button #1 is pressed, the logic tries to reprograms the FPGA with “Conf2”, using a second configuration bitstream located farther in the Configuration Flash.
- Conf2 does only display “2” on the seven segments display.
  It does not include any reprogramming logic. So it is a very trivial design indeed !

Apart from the reprogramming logic, Conf1 is also very simple.

To sum up :

- At power up, the FPGA loads the “Safe” bitstream at address 0: “Conf1” and displays “1”.
- When the user does press Button #1, the Conf1 design does send appropriate commands to the Remote Update Megafunction, asking for loading the middle configuration image : Conf2 loads.

Note that you cannot load *directly* Conf2 at power up (AS Mode), Conf1 must always be loaded first.

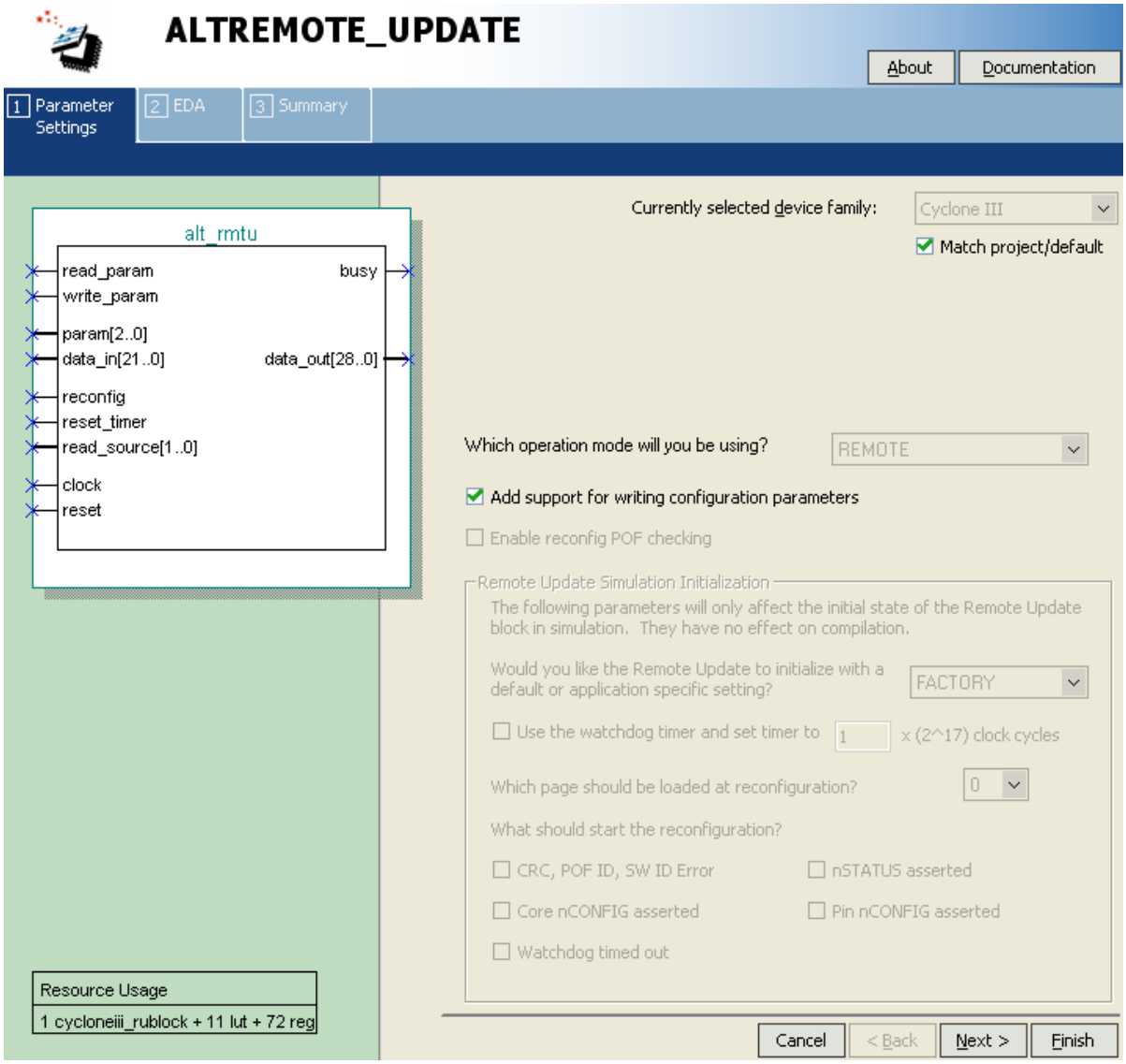
Note also that the EPCS Memory used in the DE0 board is small (4 MBytes), so two configuration bitstreams can fit only if they are compressed (you need to remember this while experimenting).
This is not a problem anymore with the Cyclone 10 LP kit.

Last Note : for the Cyclone 10 LP update, we use SystemVerilog to implement both designs.

The Multi-Configuration Project (Conf1)

This project embeds the Megafunction named ALTREMOTE_UPDATE.

For the Cyclone III Family, this macro has practically nothing to configure, just make sure that you activate the option to "*Add support for writing configuration parameters*".



On the last configuration page (Summary pane), create the vhdl "variation" file and the Instantiation template. The bsf symbol file and the component declaration are optional.

File	Description
☑ alt_rmtu.vhd	Variation file
☐ alt_rmtu.inc	AHDL Include file
☐ alt_rmtu.cmp	VHDL component declaration file
☑ alt_rmtu.bsf	Quartus II symbol file
☑ alt_rmtu_inst.vhd	Instantiation template file

Build your application and insert the *Altremote_update* Megafunction (cut & paste from _inst.v(hd)).

You will need to connect :

- a **Clock**, that we recommend to not be faster than 10 MHz. The documentation is not very clear about that but it is clearly inappropriate to use a 50 MHz !
- A **Reset**, that must be resynchronized to the clock above.
- The **Reconfig** signal, active high, which will launch the FPGA re-programming,
- **Param [2:0]** : vector to select an internal register.
- **Data_in [21:0]** : data to be written in the internal registers.
- **write_param** : to write registers in the RMTU block.
- **read_param** : if you want to read registers in the RMTU block.
- **Busy** : driven by the Megafunction, required for the handshake.

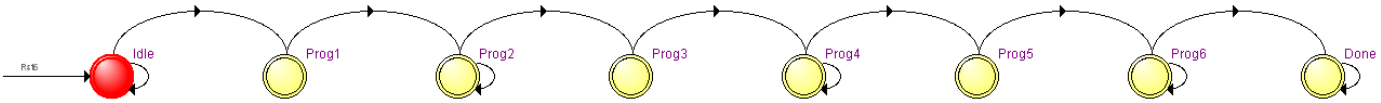
To **load** the second (user) **configuration** your need (at least) to :

1. Write the Flash Address for Conf2 **divided by 4** into register #4
   Important : the division by 4 is a bit of a mystery. In some cases (like a newly created project for Cyclone 10 LP), it must NOT be done and the un-divided address must be used !
2. Write with Data_in[0]=0 into register #3 to inhibit the Watchdog.
3. Raise the Reconfig signal (during at least 250ns).

To **write** a **Parameter**, you must comply with the following protocol :

1. Set **Param[]** to the desired register number, and set **Data_in** to the expected value.
   Except when programming the Flash Address location for the bitstream to load, not all the Data_in bits will be used.
2. Assert **Write_param** for one clock cycle.
3. By safety, wait for one extra clock cycle, then wait until Busy goes low, thus indicating that the Megafunction is able to take in another parameter.

The control of the Megafunction can be implemented quite simply using a Finite State Machine.



Note #1: For Cyclone III (and above), Altera says *"It is strongly recommended that you turn on both the **Cd_early** and **Osc_int** option bits"*. We only set Cd_early in the Cyclone III and IV projects.

Note #2: Reading a parameter is very similar. Just remember that you will also need to set the two-bits **read_source** input to the proper value (good luck with the documentation though).

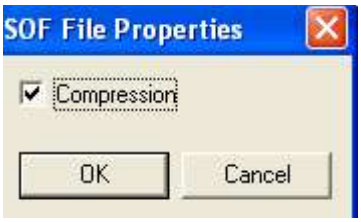
Project “Conf2” – DE0 Cyclone III kit - VHDL

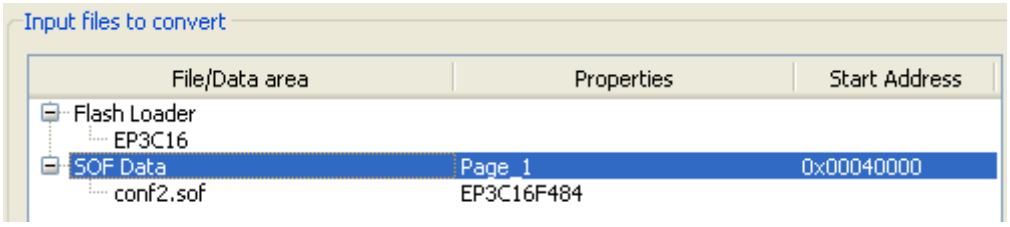
This is the top level for this trivial project :

```
-- Conf2.vhd
-- ------------------------------------------------
--   Trivial DE0 Project display "2"
-- ------------------------------------------------
-- (c) A.L.S.E.
-- Author  : Bert Cuzeau - info@alse-fr.com -  Web: http://www.alse-fr.com
--
Library IEEE;
  use IEEE.std_logic_1164.all;
-- ----------------------------------------
    Entity Conf2 is
-- ----------------------------------------
      Port (  CLK      : in  std_logic;                    -- Global Clock 50 MHz     PIN_G21
              BUTTON   : in  std_logic_vector(0 to 2); -- active High
              LEDG     : out std_logic_vector(0 to 9); -- Active High
              HEX0_D   : out std_logic_vector(0 to 6); -- Active low
              HEX0_DP  : out std_logic;                    -- Active High
              HEX1_D   : out std_logic_vector(0 to 6);
              HEX1_DP  : out std_logic;
              HEX2_D   : out std_logic_vector(0 to 6);
              HEX2_DP  : out std_logic;
              HEX3_D   : out std_logic_vector(0 to 6);
              HEX3_DP  : out std_logic                   );
end Conf2;
-- ----------------------------------------
    Architecture RTL of Conf2 is
-- ----------------------------------------
Begin
  HEX0_D <= not "1101101"; -- "2"
  HEX0_DP <= BUTTON(0);
  LEDG   <= (others=>'0');
  HEX1_D <= (others=>'1'); HEX1_DP <= '1'; -- all Off
  HEX2_D <= (others=>'1'); HEX2_DP <= '1';
  HEX3_D <= (others=>'1'); HEX3_DP <= '1';
end architecture RTL;
```

Create the project using just the above module, **assign the pins** (we provide a script), create the appropriate SDC timing script, and compile the project. This creates the SOF file.
Now under Quartus, proceed exactly as follows:

- File > **Convert Programming File.**
- Programming File Type : **JIC** (JTag Indirect Configuration File)
- Configuration Device : **EPCS4**
- File name : **Conf2.jic**
- Flash Loader : Add Device = **Cyclone III – EP3C16**.
- SOF Data Properties : **Page_1** ,
  Address mode = **Start**,
  Start Address (32 bits) = **0x40000** (four zeroes)
- Add File : **Conf2.sof**
- Highlight Conf2.sof – Properties, **activate Compression** !





- **Generate** to create the JIC file.
- **Program** the DE0 using the JIC file.
- Turn the board off then on again: nothing should change but don't worry if your board does not “boot” anymore: the programming may have overwritten the existing default bitstream. But you now have your Conf2 bitstream in the middle of the Flash.

Project Conf1

This project has all the “complexity”.

The different functions located in the top level are :

- **Power On Reset** Module instantiation. Works from the external clock and produces a reset pulse. We add this module to all our designs, so we don't have to rely on power up values nor on any external reset input.

```
POR_i: entity work.POR port map (CLK,RST); -- Very simple Power On Reset
```

- A simple C**lock divider** (Johnson counter) to divide the external 50 MHz clock into a local 6.25 MHz clock named CLK6. All the logic (except indeed the divider) is synchronous to this derived clock.

```
-- Divide the external 50 MHz clock by 8.
Johnson <= x"0" when Rst='1'
      else Johnson(1 to 3) & not Johnson(0) when rising_edge(CLK);
CLK6 <= Johnson(0);  -- CLK6 is then a derived clock at 6.25 MHz
```

- **Resynchronization** of the Buttons (especially Button2) and of the local Reset.
  Note that two flip flops would be better, but this is just a demonstrator.

```
BUTTONS <= not BUTTON when rising_edge(CLK6);
RST6 <= '1' when RST='1' else '0' when rising_edge (CLK6);
```

- **Blinker** instantiation to drive HEX3_DP (blinking decimal point showing some activity and proving that the internal clock is functional).

```
-- LED Toggles every second
Blink_i: entity work.Blink port map (Clk6,Rst6,HEX3_DP);
```

- Discrete and Seven-Segment LED drivers (most are OFF).

```
-- LEDs
LEDG    <= (others=>'0');
HEX1_D  <= (others=>'1'); -- Off
HEX2_D  <= (others=>'1'); -- Off
HEX3_D  <= (others=>'1'); -- Off
HEX0_D  <= not "0110000"; -- seven-seg "1"
HEX0_DP <= not BUTTONS(0);-- copies push button 0
HEX1_DP <= '1';           -- Off
```

- The **altremote_update Macro** instance:

```
-- Instantiate the Remote Update Configuration block
rmtu_i: entity work.alt_rmtu PORT MAP (
    clock        => CLK6,                 -- 6MHz clock
    reset        => RST6,                 -- Reset
    data_in      => data_in(21 DOWNTO 0), -- Parameter input data
    param        => param,                -- Parameter address
    write_param  => write_param,          -- Write Enable
    busy         => busy,                 -- no write allowed while busy
    reconfig     => reconfig,             -- Launches the reconfiguration
    read_source  => "00",                 -- not used
    read_param   => '0',                  -- not used
    reset_timer  => '0',                  -- not used
    data_out     => open  -- data_out(28 downto 0) -- not used.
);
```

- Note that even if the projects are quite trivial and pose no timing challenge, we have update all the projects with complete **SDC timing constraints** files.

- The **State Machine controller** to program the RMTU and launch the reconfiguration when requested.

The purpose of this version is to be the implementation as simple as possible while building a very robust and easily maintainable code.

See the code next page.

```
-- Reconfiguration State Machine (6 MHz clock)
process (Clk6,Rst6)   begin
  if Rst6='1' then
    HEX2_DP <= '1';  data_in <= x"000000"; param <= (others=>'0');
    reconfig <= '0'; write_param <= '0'; State <= Idle;
  elsif rising_edge(Clk6) then
    write_param <= '0';  -- we need one clock cycle pulses only
    case State is

      when Idle =>
        param <= "100";       -- Boot Address
        data_in <= x"010000"; -- 0x40000 / 4  BEWARE ! Sometimes NOT necessary to /4
        if BUTTONS(1)='1' then
          write_param <= '1';
          State <= Prog1;
        end if;

      when Prog1 =>
       State <= Prog2;
      when Prog2 =>
        param <= "011";       -- Watchdog Enable register
        data_in <= x"000000"; -- bit 0 = 0 : WD is disabled
        if Busy='0' then
          write_param <= '1';
          State <= Prog3;
        end if;

      when Prog3 =>
       State <= Prog4;
      when Prog4 =>
        param <= "001";       -- Write Early ConfDone Check bit
        data_in <= x"000001"; -- bit 0 : Early ConfDone
        if Busy='0' then
          write_param <= '1';
          State <= Prog5;
        end if;

      when Prog5 =>
       State <= Prog6;
      when Prog6 =>
        if Busy='0' then
          State <= Done;
        end if;

      when Done =>  -- DEAD END !
        Reconfig <= '1';

    end case;
  end if;
end process;
```

Notes :

- In register #4, we had to program the Flash address **divided by 4**. However, **in some other cases** (like the Cyclone10LP project)**, the address should not be divided !**
  We think /4 is needed if the RMTU block was generated in 2011 or around.
- When we tried first without disabling the Watchdog, we noticed that it was active by default in the loaded configuration. Symptom: Conf2 always reverted automatically to Conf1 after a variable amount of time (around 30 seconds). If Conf2 does not implement the remote configuration logic, you must deactivate the Watchdog before launching the reconfiguration.
  See next page for using the Watchdog.
- We leave Reconfig set in state Done (which is a dead end) since the FPGA will “suicide” immediately and will try to load Conf2. There is no need to clear this bit.
- In Active Serial mode, the “Page” information is not relevant (no Pgm[2:0] pins).
  This information (and the Pgm pins) are used for Parallel programming devices.
- We comply with Altera's recommendation to set the “Early ConfDone bit” for what is worth. We don’t think it’s particularly useful though (for Cyclone III=10LP and IV). It might change the value of the error status returned in case of configuration failure (due to an incorrect image).

Experimenting

Note: If, after experimenting, you want to reprogram your DE0 board as it was delivered, the archive should include the default bitstream. If it doesn't, you can find this default bitstream and default design on the DE0 CD-Rom or on the Terasic Web site.

- After having assigned the pins correctly (and created the proper SDC timing script), compile the project to generate an SOF programming file.
- Proceed with “Convert Programming File” like for Conf2, but leave the Start address by default (will be 0) and produce a compressed bitstream named **Conf1.JIC**. Program the board with this new file. Note that it is possible to build a single JIC file containing both bitstreams in one file.
- Power down the board, then up again: the display should show “1” and DP2 should be blinking.
- Press the middle button BUTTON1: the board should reprogram itself and load Conf2 as witnessed on the LED.

Good ! Our “dual boot” system works as expected. Let's try a few more things :

- Destroy the bitstream image for Conf2. You can for example build a compressed JIC using a Start Address of 0x3FF00 instead of 0x40000, and program it (this bitstream is in the archive).
- Now, when you press button2 after power up (while in Conf1), you'll see that the device keeps trying unsuccessfully until you release Button1, then Conf1 indeed loads again correctly. In some cases, this may not be what you need, especially if Conf1 is supposed to provide a safe mechanism to update Conf2 remotely. The next chapter below is just about this !

SDC Timing Constraints File

In general and for the Safe image in particular, you should build a correct SDC constraints file.

As we have seen, we have the main 50 MHz clock, a 6.25 Mhz derived clock generated by the Johnson counter, and false paths for the I/Os.

The Cyclone IV and Cyclone 10 LP projects in the archive have correct SDC files.

Example :

```
# ------------------------------------------------
# SDC constraints for Remote Update "Safe" project
# ------------------------------------------------
# (c) ALSE - B. Cuzeau
create_clock -name Clock -period 20 [get_ports CLK]
derive_clock_uncertainty
create_generated_clock -name CLK6 -divide_by 8 -source [get_ports CLK] [get_pins
{Johnson[0]|q}]
set_false_path -from [get_ports BUTT*]
set_false_path -from [get_ports SW*]
set_false_path -to   [get_ports LEDS*]
```

Cyclone IV – BeMicro SDK
Using the Watchdog

Projects : **SafeWD** and **UserWD**

In the previous example, we disabled the Watchdog.
If we leave the watchdog Enabled, we can only load User designs that:

1. Implement the Remote Update block (“RMTU”)
2. Continuously pulse the reset_timer input of the RMTU.

Otherwise, the User configuration will “suicide” after the watchdog delay programmed by the Safe project, and the “Safe” configuration will be automatically reloaded.

For a change, we have implemented this example on the Arrow **BeMicro SDK kit** which uses a Cyclone IV FPGA, which is still supported by the latest Quartus versions (2023). **In the 2023 archive that we provide, the project has been updated to Quartus 22.1.**

SafeWD project

The difference with Conf1 is that we Enable the Watchdog and we load the WD timer period register with 512 (0x200). Knowing that he WD counter has 17 Least Significant Bits, the period will be 512 * 2^{17} internal clock cycles. With an 8 MHz clock, this would correspond to ~8 seconds. We adopted this value since it leaves time to observe the sequence and to rock a switch. In practice, you may wish to adopt much shorter delays, but you need to leave enough time for the User configuration to load and for your internal Power On Reset (if any) to end. ***Reminder***: the watchdog runs on its own **internal** clock !

Note that even with Quartus 22.1, it is still necessary to divide the User address by 4 !

The modified code is :

```
      when Prog2 =>
        param <= "010";       -- Watchdog Period Load
        data_in <= x"000200"; -- 512 * 2**17 -> ~10 seconds
        if Busy='0' then
          write_param <= '1';
          State <= Prog2a;
        end if;

      when Prog2a =>
        State <= Prog2b;

      when Prog2b =>
        param <= "011";       -- Watchdog Enable register
        data_in <= x"000001"; -- only bit 0 is used (WatchDog enable).
        if Busy='0' then
          write_param <= '1';
          State <= Prog3;
        end if;
```

UserWD project

The difference with Conf2 is that we must instantiate a Remote_update Macro-function in the User design. If the DIP-Switch SW1-1 is in OFF position, the reset_timer input is pulsed continuously :

```
-- Generate the WD clear (Clock divider) if SW1-1 is up
WDgen <= (others=>'0') when SW(1)='0' else WDgen + 1 when rising_edge(CLK);

-- Instantiate the Remote Update Configuration block & pulse reset_timer
rmtu_i: entity work.alt_rmtu PORT MAP (
    clock       => '0',                  -- not used !
    reset       => '0',                  -- not used !
    .../...                              -- not used ports
    reset_timer => WDGEN(WDGEN'high),    -- a pulse does clear the WD
    data_out    => open               ); -- not used.
```

Experimenting on the BeMicro SDK

- Program the BeMicro II (sdk) with the two JIC files.
- Set SW1-1 (left Dip switch) to the ON position
- Power down, then up. Or use the REFCG button : this load SafeWD.
  LED1 is ON and LED8 does blink. (LED7 is on)
- Press the “User” button (on the right) : this loads UserWD
  LED3 is ON
- Wait for ~8 seconds : you will see that SafeWD is automatically re-loaded, due to the Watchdog. This demonstrates that the internal clock is close to 8 MHz.
  Noe that if you set SW1-1 OFF before the WD expires, or if it is OFF initially, then the FPGA will remain in User mode.
- Make sure SW1-1 is in OFF position
- From SafeWD, press the “User” button.
  This loads UserWD, and you will not return to SafeWD because the Watchdog is continuously reset

Note that you can create a single JIC file that contains *both* bitstreams (see **Safe_and_User**.cof).

Cyclone 10 LP Evaluation Kit

There are few kits available for the Cyclone 10 LP family.
The Evaluation kit that we use here has been designed and is sold by Intel (and the usual resellers including Mouser, Digikey and Terasic). The kit documentation is available from the Intel website.

It's a very interesting kit with its Arduino + PMOD + HE10-40 extension connectors, Gigabit Ethernet Phy, HyperRam memory, external PLL and onboard JTag blaster. Its price just under $100 is also a bonus.
For the Remote update demo, its downside is the absence of 7-seg display.

Unfortunately, as of Jan 2023, this kit is completely UNavailable and nobody seems to know when it will be available again (if ever).

Even if you don't have this kit, this Appendix is interesting since you can apply it to *any* Cyclone 10 LP board that you may have, and it is based on the latest version of Quartus Prime Std (as of Jan 2023).

We won't repeat what has been described above in this ApNote. We'll focus on the differences due to using the latest version of the IP and of the tools.

Conf2 : simplest Blinker project

This is a trivial project that does blink the two right LEDs. There is nothing special in this project since the design doesn't use nor instantiate the remote update IP.

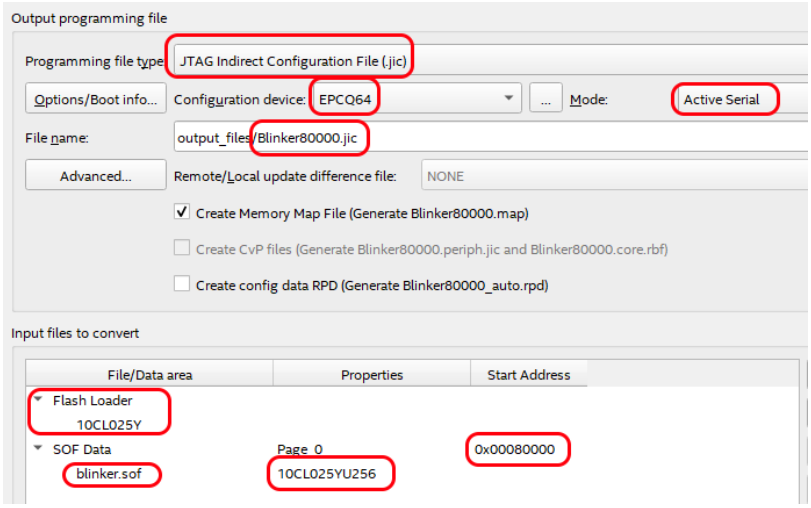
The code is just 3 lines and uses 5 pins :

```
// ----------------------------------------
//  Blinker design for remote update ApNote
// ----------------------------------------
module blinker (input logic Clk50M, output logic [3:0] LED ); // E1,J13,J14,K15,L14
logic [25:0] Cntr;
always_ff @(posedge Clk50M) Cntr++;
assign LED = {2'b11,Cntr[25:24]}; // LEDs are active low
endmodule
```

The device is 10CL025YU256I7G and the SDC is quite trivial.

Compile the Blink project to generate the blink.sof programming file. Activate the compression option. You can test by downloading the FPGA to see the LEDs blinking.

Use "Convert Programming File" to convert into a standalone JTag Indirect Configuration (JIC) file while ensuring that this design will be programmed at address **0x80000** in the Flash :



Note : we have prepared Blinker80000.cof that you can load to set up the parameters as above.

Program the Flash. At this stage, there is no way to load the FPGA with this configuration from Flash.

Conf1 Project : using the Remote udpate IP

This is where things become more interesting ! We will be able to load the FPGA with the Blinker configuration that we have programmed in the Flash memory in the previous steps.

Project creation

You can start from the Blinker project and create an extended project (same FPGA, same Flash, same 5 pins). You must add the five push buttons : PB[3:0] + Resetn (we actually only use PB[3] and Resetn).

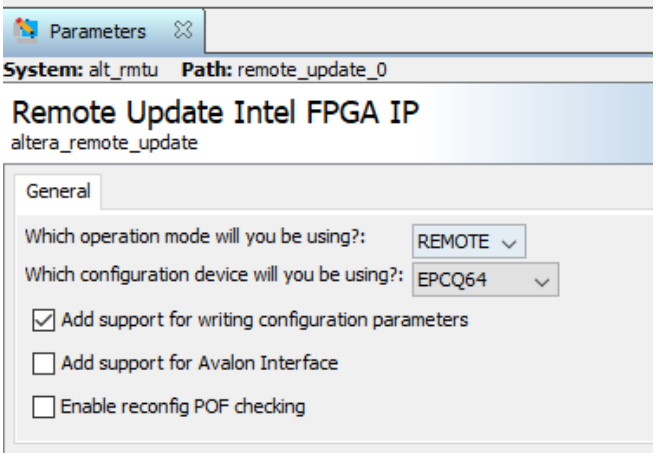
As seen in the original Cyclone III project, you must :
- Create a slower clock (like 6.25 MHz)
- Resynchronize the push button.
- Generate and instantiate the Remote Update IP (see below)
- Create the State Machine to control the IP and load the Blinker configuration when the button is pressed.

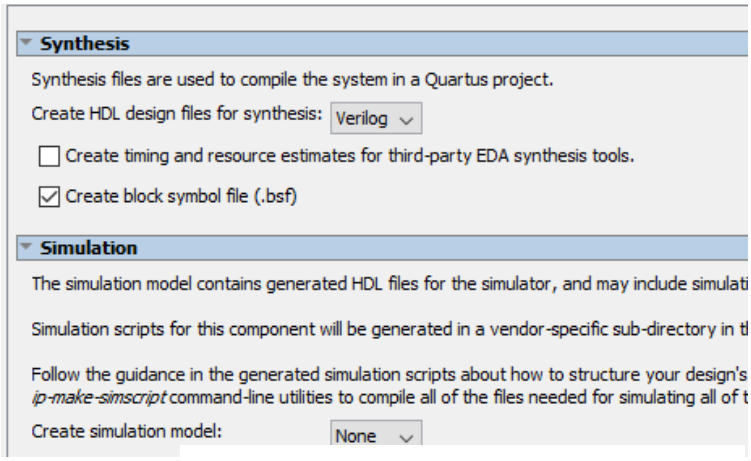
To create a good base for adding features later to this project, you could also add :
- a Tick generator
- a POR

Remote Update IP

For the Cyclone 10 LP kit update, we use the IP Catalog and Platform Designer.

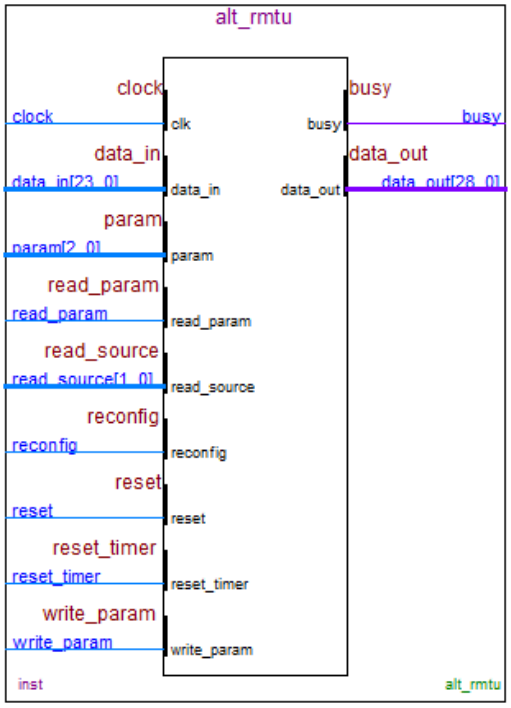
Select **EPCQ64** as configuration device, and **Verilog** for language :





Here is the symbol generated :

You must add the generated QIP file to your project.



Reconfiguration State Machine

You can find the complete SystemVerilog code for Conf1 in the archive.

This code does also implement read transactions from the Remote Update IP.

In the comments, you will find the interpretation of the 5 bits status that is read :

```
Cyclone III/IV Series Status (Param "111") :
Diag(4) = nconfig_source : External configuration (nCONFIG) assertion
Diag(3) = crcerror_source: CRC error during application configuration
Diag(2) = nstatus_source : nSTATUS asserted by external device as result of an error
Diag(1) = wdtimer_source : User Watchdog Timer timeout
Diag(0) = runconfig_source: configuration triggered from logic array
```

In the code, the status is read and not used, but you can easily enhance the code to do different things depending on the reconfiguration reason above.

The sequence is similar to the original VHDL example, but we have inserted a 1 second delay before turning on the LEDs and checking for the Button3 press.

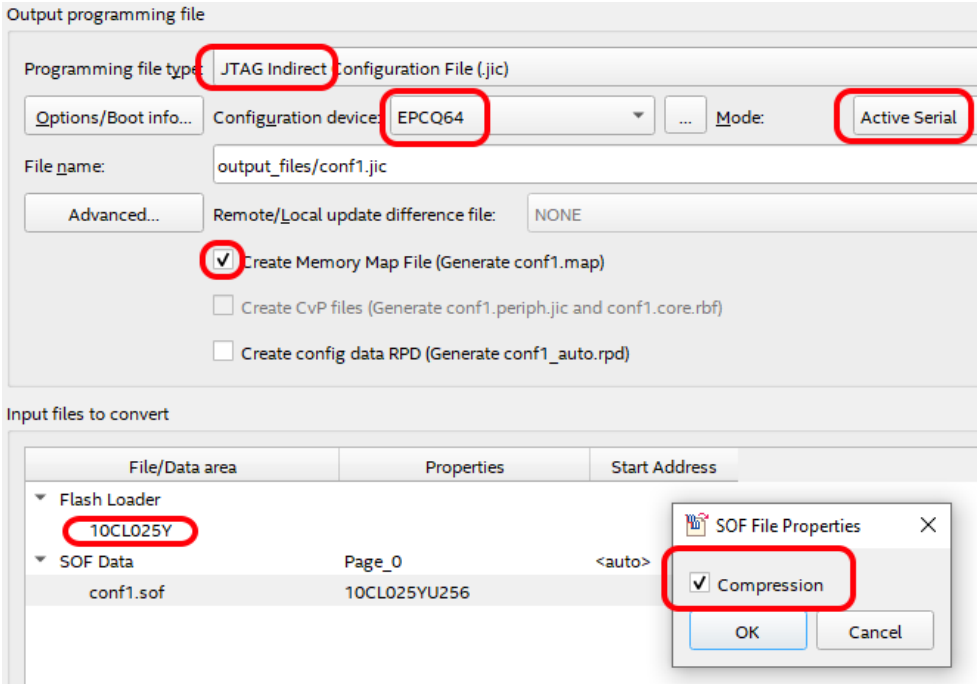
We don't forget to disable the Watchdog !

We use the byte address in the Flash memory, without division by 4.

Programming Conf1

You can download (conf1.sof) and verify the behavior as described next page. But the way to go is to prohgram conf1 in the Flash memory.

Use the same steps as for Conf2 to generate the JIC file, but you can leave the base address to <auto> or set it manually to be 0x0000 :



Note that, for convenience, we have prepared a .cof file that you can load.

After generating the JIC file, you can take a look at the conf1.map file :

```
BLOCK      START ADDRESS     END ADDRESS

Page_0     0x00000000        0x00032CB6

Configuration device: 10CL025Y
Configuration mode: Active Serial
```

With conf1 starting at 0x80000, the compressed image for conf1 takes less than half the space.

Use the programmer to prom conf1. It will not erase conf2, but there is a command to erase the whole Flash memory in the programmer.

Testing on the kit

- Power down (remove the USB cable).
- Power up again the board → conf1 is loaded, as shown by the two leftmost LEDs on.
- Press the leftmost button (3) → conf1 is loaded and you see the two right green LEDs flashing

We can return to conf1 :

- Press the rightmost button → the FPGA is reconfigured, so conf1 is loaded
- Press PB3 → conf1, just as above.

Securing the Reconfiguration

IMPORTANT: for some FPGA families (like Arria II gx, Cyclone V and many others), the attempt to load an incorrect User configuration bitstream will **not** return with an error ! In this case you must check the configuration data BEFORE attempting the configuration. There is an option available for this in the Megafunction (at the cost of some extra logic and configuration time). The Cyclone III & Cyclone IV (and therefore the Cyclone 10 LP) families are not affected by this problem.

In the cases above and in many applications, we need `Conf1` to aware that there was a *prior attempt* to load `Conf2` that *failed*, in which case it should not launch the reconfiguration, but should probably signal the error and later try to reprogram the User configuration image in the Flash memory.

This is possible by reading the internal registers in the `altremote_update` Macro-function, but the mechanism and the implementation seem to be somewhat FPGA dependent.
We only looked at Cyclone III (Cyclone 10 LP) and Cyclone IVe for this ApNote.
Moreover, by reading properly some registers, you can (at least in Cyclone III) get independent information about the **two last previous** attempts *before* the one that indeed succeeded (but Intel says *"but only for debugging purposes"* !). If you focus on the oldest status, you will de-facto implement a two-retries scheme, because you'll decide to stop trying after two unsuccessful reconfiguration attempts.

However, the documentation is again quite difficult to understand... Our experience gave us some working code: reading from **Param="111"** with **read_source="01"**, but the value returned is not trivial to interpret. You need the table below.

Status Register bits (5 bits binary vector) and reconfiguration reason :

Code	Reason of configuration
10000	external nCONFIG source
01000	CRC error source
00100	nSTATUS source
00010	User watchdog timer source
00001	core nCONFIG source

You can try this design using with `Safe.jic` (make sure you completely Erase the Flash memory first to prevent Conf2 from loading). "Safe" has a one second delay before reading the button and attempting the reconfiguration, and the 5-bits vector read back is displayed on the second digit (and a decimal point), just after an "E" (like Error).

It is easy to modify the code and the FSM to implement register read & test. You will need to use `read_source`, `read_param`, and `data_out`. Use `read_param` just like `write_param` (with the busy flag).

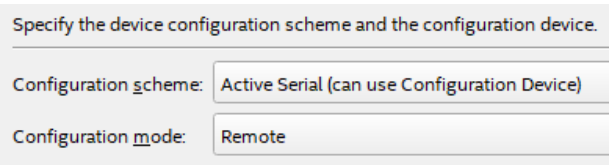
We suggest you also add some delay between power up and the reconfiguration attempt, to display the read back status (we used the Hex display, but you could also send characters over a UART).

If you are dealing with another FPGA family than Cyclone III, Cyclone IV or Cyclone 10 LP, or if you use other reconfiguration method than AS, you can still apply the methodology and start from this code.

Last : if Conf2 has the ability to launch a reconfiguration to Conf1, then Conf1 would need to read the last load address to recognize the situation (since the previous User config was successful !). This is possible.

Thinks to remember

- Use a **slow enough clock** to drive the Mega function. If you don't, TimeQuest might report the problem. Wasting a PLL for this might not be the best choice (takes some time to get locked, you need to handle the LOCKED signal appropriately etc etc), especially if you want to launch the reconfiguration as soon as possible after power up.
  A simple Johnson counter as in the example provided can do the job.
- If you do not use the Remote update function in the “User” configuration, make sure you **inactivate** the **Watchdog** before launching the reconfiguration.
- If you enable the **Watchdog**, all the User projects that you load must : (1) instantiate a remote_update macro and (2) pulse *continuously* the reset_timer input of the rmtu block.
  Note: the watchdog is insensitive to the rmtu’s reset input (it will reconfig even if reset is forced).
- Typically, the Safe image should **read and test the reconfig Status** as shown, and only attempt loading the User image if the status is 00001 (binary).
- We think the **division by 4** of the image address is only required if the RMTU block has been generated by an old version of Quartus (at least until version 11.1, maybe a bit after).
- Whenever possible (device family, memory used, configuration method...), activate the **bitstream compression**. Otherwise, you might not be able to fit several bitstreams in the configuration device.
- Reminder (using Conf1 and Conf2 naming convention): **in AS mode, you must always start in the default configuration** (Conf1), then load the User configuration (Conf2) from Conf1.
  If this launch (from Conf1) is attempted automatically, you should first verify that your are not in Conf1 due to a failure while loading Conf2 ! (in which case you want to stay in Conf1). To avoid the infinite retry, you should implement the *parameter read* function in your state machine, and read the appropriate internal register(s) before attempting the reconfiguration as explained above.
- **Practical experiments** are necessary since simulation of reconfiguration error is not possible and using an ELA (SignalTap) is difficult -it dies at reprogramming-.
- **Important** : if you use the remote update feature, you should enable the option located in:
  Device > Pin & Options > Configuration > Configuration mode = “`Remote`”.
  Indeed, if you change your mind and remove the megafunction, you'll need to revert to “Standard”, otherwise the fitter might complain. Whatever your FPGA family, this option seems to be:
  `set_global_assignment -name STRATIXIII_UPDATE_MODE REMOTE` (or `STANDARD`)



- When you program one bitstream, make sure you **do not erase nor overlap** the other !
- Do **not** use the **RBF** programming files for active serial devices, this wouldn't work.

For other families !

- Around the V families a serious regression was introduced ! In case of an incorrect User bitstream, the return to the Safe image was not guaranteed, even if the watchdog was activated. A partial mitigation was introduced that tries to verify the image as much as possible before loading.
  If you are using the affected families, you must take this into account.
- For SOC families, you will find some Reference Designs implementing Remote Update.

Archive is available

We recommend that you follow the indications and build your own application using `Altremote_update`.

But in case you want to experiment and start from our examples, you can download the new updated and complete archive "RemoteUpdateFiles.zip" which does contain various Quartus Projects archives :

CycloneIII/CycloneIII.zip	The original Cyclone III projects (for DE0), that can be recompiled and tested using Quartus 13.1 (the latest version supporting the Cyclone III family). It does also include the DE0 factory default project.
CycloneIV/CycloneIV_conf1.qar	The updated BeMicro SDK Safe project updated to Quartus 22.1
CycloneIV/CycloneIV_conf2.qar	The updated BeMicro SDK User project updated to Quartus 22.1
Cyclone10LP/Cy10LP_conf1.qar	The new Cyclone 10 LP evaluation kit Safe project
Cyclone10LP/Cy10LP_conf2.qar	The new Cyclone 10 LP evaluation kit User project

Conclusion

As you can guess, I didn't build this Application Note to receive questions nor to replace Intel's Technical Support ! But hopefully it will help you figure out more easily how to set up and use a multi-configuration system.

I have updated it in 2023 for the Cyclone IV project and I added a Cyclone 10 LP project, both using the latest version of Quartus, but the disclaimer notice page 1 still applies.

Keep in mind that we have off-the-shelf **solutions for Remote Flash Programming through Ethernet** or through other links like **RS232 or Modbus**.

Bertrand Cuzeau – Chief Technology Officer A.L.S.E

-=oOo=-