Secure IP Flow for IP Vendors and Libero SoC User Guide

Libero SoC v2025.1

Introduction (Ask a Question)

Microchip has adopted IEEE® 1735-2014 and supports an encrypted IP design flow for the SmartFusion® 2, IGLOO® 2, RTG4™, PolarFire®, and PolarFire SoC silicon families.

- See Securing Your IP Core section to secure your IP core.
- See Running Libero SoC with Encrypted IP section for information on running Libero® SoC with encrypted IP.

Together with its OEM tools, Synplify Pro® from Synopsys® (Synplify Pro ME I2013.09MSP1 or later), ModelSim® (ModelSim 10.2c or later) from Siemens (both of which support IEEE 1735-2014), and Libero SoC (v11.3 or later) enable a seamless design flow for designers targeting SmartFusion 2, IGLOO 2, RTG4, PolarFire, and PolarFire SoC when they use encrypted IP cores in their design.

The use of IP cores not only shortens the design cycle time but also provides proven and reliable design components for reuse in multiple applications. Using an IP core in the EDA design flow involves two conflicting considerations that must be resolved: IP security and IP interoperability across different EDA tools.

Libero SoC Secure IP Design Flow Requirements (Ask a Question)

The following table lists the software and hardware requirements for Designing with Secure IPs in Libero SoC.

Table 1. Software and Hardware Design Requirements

Design Requirements	Description
Hardware Requirements	
PolarFire®, PolarFire SoC, RTG4™, and SmartFusion® 2/IGLOO 2® Family devices	This feature is supported for the PolarFire, PolarFire SoC, RTG4, and SmartFusion 2/ IGLOO 2 family devices.
Host PC or Laptop	Windows® 64-bit Operating System (OS)/Linux® 64-bit OS.
Software Requirements	
Libero® System-on-Chip (SoC)	v11.3 or later.
Synplify version	Synplify Pro® ME I2013.09MSP1 or later.
ModelSim® version	ModelSim 10.2c or later.
Encryption Script Requirements	
OpenSSL	Most Linux and Cygwin have OpenSSL pre-installed. Provide OpenSSL installation location in the “PATH” environment variable of the system.
Perl	Any version of Perl with the following packages installed: FindBin, Math, Getopt, File, and MIME.
Cygwin (for Windows OS)	For executing Perl Script on Windows OS.
Public Keys for encryption	See section Public Key from EDA Vendors.

IP Security and IP Interoperability Across Design Tools (Ask a Question)

The use of IP cores not only shortens the design cycle time but also provides proven and reliable design components for reuse in multiple applications. Using an IP core in the EDA design flow involves two conflicting considerations that must be resolved: IP security and IP interoperability across different EDA tools.

Table of Contents

Introduction....1
Libero SoC Secure IP Design Flow Requirements....1
IP Security and IP Interoperability Across Design Tools....2

1. Encryption and Decryption....4
1.1. IEEE 1735-2014 Standards for IP and EDA Vendors....4
1.2. Encryption Algorithms....5
1.3. Encryption Envelopes....6
1.4. Decryption Envelopes....7

2. Securing Your IP Core....9
2.1. Encryption of IP Core with IEEE 1735-2014 Scheme....9
2.2. Public Key from EDA Vendors....10
2.3. Adding an Encryption Envelope to Your RTL....10
2.4. encryptP1735.pl Script....11
2.5. Packaging and Bundling the Encrypted IP and the Data Key....15

3. Running Libero SoC with Encrypted IP....16
3.1. To run Libero SoC with Encrypted IP....16
3.2. Encrypted IP Design Flow Must Use Verilog Netlist from Synthesis....16
3.3. Import Encrypted IP Core as HDL....17
3.4. Run Synthesis....19
3.5. Run ModelSim Simulation....20
3.6. Libero SoC and Encrypted IPs....21

4. Frequently Asked Questions....26

5. Revision History....27

Microchip FPGA Support....28

Microchip Information....28
Trademarks....28
Legal Notice....28
Microchip Devices Code Protection Feature....29

1. Encryption and Decryption (Ask a Question)

The following sections provide information about encryption and decryption to consider when designing an encrypted IP core.

1.1. IEEE 1735-2014 Standards for IP and EDA Vendors (Ask a Question)

IEEE 1735-2014 is an encryption scheme proposal adopted by most IP and EDA vendors to ensure the interoperability of IP cores among the IP vendors and EDA tools. The objective of IEEE 1735-2014 is to serve the IP vendors and the EDA community in the following ways:

- For the IP vendor: Protect the security of the IP core in the design flow across different EDA tools.
- For the IP core users and EDA tool vendors: Ensure the interoperability of the IP core across different EDA tools.

1.2. Encryption Algorithms (Ask a Question)

Libero SoC supports the following encryption algorithms:

- des-cbc
- 3des-cbc
- aes128-cbc
- aes256-cbc

There are two major classes of encryption methodologies: Symmetric and Asymmetric.

1.2.1. Symmetric Encryption (Ask a Question)

This encryption scheme uses a special string as a key to encrypt data. The same key is used to decrypt data. See Figure 1-1.

Examples of this type of encryption algorithm include:

- Data Encryption Standard (DES), such as des-cbc.
  - Triple Data Encryption Algorithm (TDES), which uses the DES algorithm three times, such as 3des-cbc.
- Advanced Encryption Standard (AES), such as aes128-cbc and aes256-cbc.

1.2.2. Asymmetric Encryption (Ask a Question)

This encryption scheme uses two different keys: one for encryption and another for decryption. The end user generates two keys, one public and another private. The end user distributes the public key to whoever needs it for encryption and keeps the private key to use for decryption (see Figure 1-2).

Common examples of asymmetric encryption algorithms are:

- Diffie-Hellman (DH)
- Rivest–Shamir–Adleman (RSA)

1.2.2.1. Two Levels of Encryption (Ask a Question)

There are two levels of encryption when producing an encrypted IP core. Figure 1-1 shows the first level of encryption, where the IP core vendor uses a session (random) key to encrypt the IP content. Figure 1-2 shows the second level of encryption, where the IP core vendor uses the public keys from EDA vendors to encrypt the session key.

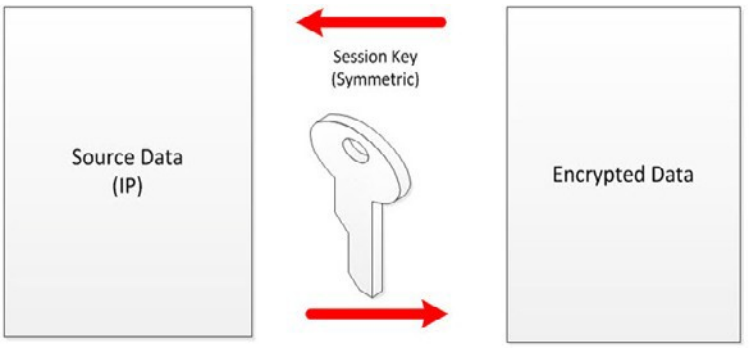
A public key must be provided for each EDA tool to the IP core vendor.

For Libero SoC customers who use third-party IPs in their design, the EDA vendors are:

- Synopsys for Synplify Pro
- Siemens for ModelSim
- Microchip for Libero SoC

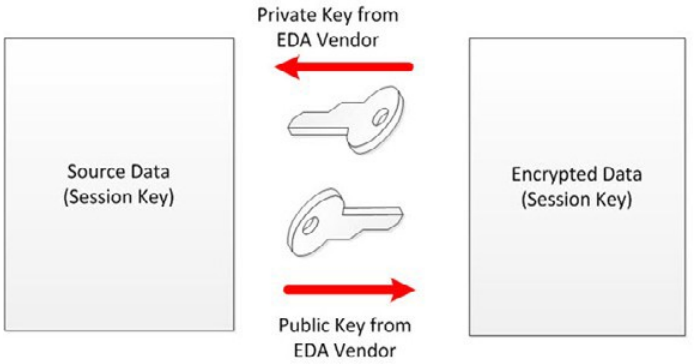
The result of the first level of encryption is the encrypted data block. The Random session key required for symmetric encryption of the data block is generated by the `encryptP1735.pl` script.

Figure 1-1. Data Encryption of Source Data IP



The result of the second level of encryption is the encrypted session key.

Figure 1-2. Session Key Encryption



1.3. Encryption Envelopes (Ask a Question)

The Encryption envelope is the preamble to the IP in the HDL file. The IP core vendor must prepare an Encryption envelope for all EDA tools, which are used with the IP. The encryption envelope consists of pragma keywords (see section Pragma Keywords) that provide the following information:

- Encryption version
- Encoding type
- Encryption agent
- Key owner
- Key name
- Key method

Following is an example of an Encryption envelope.

```
module secret (a, b, sum, clk, rstn); input[7:0]a, b;
input clk, rstn; output[8:0]sum; reg[8:0]sum;
`pragma protect version=1
`pragma protect encoding=(enctype="base64")
`pragma protect author="author-a", author_info="author-a-details"
`pragma protect encrypt_agent="encryptP1735.pl", encrypt_agent_info="Synplify
encryption scripts"
```

```
`pragma protect
key_keyowner="Synplicity",key_keyname="SYNP05_001",key_method="rsa",key_block

`pragma protect key_keyowner="Mentor Graphics Corporation",key_keyname="MGC-
VERIF-SIM- RSA-1",key_method="rsa",key_block

`pragma protect key_keyowner="Microsemi Corporation",key_keyname="MSC-IP-KEY-
RSA",key_method="rsa",key_block

`pragma protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip",
data_method="aes128-cbc"

`pragma protect begin

always @(posedge clk or negedge rstn) begin if (!rstn)

sum <= 9'b0; else

sum <= a + b; end

`pragma protect end endmodule
```

Note: The encryption envelope identifies three EDA tool vendors/key owners.

1.4. Decryption Envelopes (Ask a Question)

The Decryption envelope is the preamble to the encrypted IP. The Decryption envelope consists of pragma keywords (see section Pragma Keywords) that provide the following information:

- Encryption version
- Encoding type
- Encryption agent
- Key owner
- Key name
- Key method

Following is a Verilog example of a Decryption envelope.

```
module secret (a, b, sum, clk, rstn); input[7:0]a, b;

input clk, rstn; output[8:0]sum; reg[8:0]sum;

`pragma protect begin_protected

`pragma protect version=1

`pragma protect author="author-a", author_info="author-a-details"

`pragma protect encrypt_agent="encryptP1735.pl", encrypt_agent_info="Synplify
encryption scripts"

`pragma protect key_keyowner="Synplicity", key_keyname="SYNP05_001",
key_method="rsa"

`pragma protect encoding=(enctype="base64", line_length=76, bytes=256)

`pragma protect key_block
NfR8W3gmxwh3Bj4QxA+Qi+BhD1CTnQv7KO4UGOOS27KzF4jtejZxAewyFaShFSqRn9tRNx+u7Ivw
1m2BydGyW7MAQx2ePgbrKQbRLaN8XF/iiUFUX0QXnWDZrxtgcVHULOsPXpwd25wNyeWQkTekAsln
ubKiFDfNySxaP5W3SboZE0pMLqH+mpZlcvKljlE30uOAQQLjECEBGj1KxMZQ2hhUKLrXz34+9p68
tVzbM/u1TbsXvdPcN23UItAxNPSH5ND75rAviq7ACIVawH87/m2RshSDSVcmz7ndMpSJRQOFe2pd
usuHdCFJm1YaEaCZYfqReV7RjCzbV48d3LPtoA==

`pragma protect key_keyowner="Mentor Graphics Corporation", key_keyname="MGC-
VERIF-SIM- RSA-1", key_method="rsa"
```

```
`pragma protect encoding=(enctype="base64", line_length=76, bytes=128)

`pragma protect key_block boN+vsIsOJ/
Ihy7BF0MM2ZdaeYl2zoepUP9xdDVnlME3q5lgqZtPjMtPqTQDvwbree7NngmOUGVm WbggEEW/
UWYWajwld641fsggKfu7kcFcMhLLBu0WHUVFvQjRhdiqcBWbEKM39O0SCYTJnhQFPs0B
RZgdCwOPvZ4IEAUqx4U=

`pragma protect key_keyowner="Microsemi Corporation", key_keyname="MSC-IP-
KEY-RSA", key_method="rsa"

`pragma protect encoding=(enctype="base64", line_length=76, bytes=960)

`pragma protect key_block
MIID4jANBgkqhkiG9w0BAQEFAAOCA88AMIIDygKCA8EAxvOR7+3o0rtdoggobQ7e
3LQ5Bhjfcudafujkinm+213ui89cvxjkaYKRDadsklgfklDGTFyiYUIKasKv3MrW
xbaIlfktti2lBBdU/SDV83mLYKzAqe20/SaZR5FAZH8cyuUPxYOviHQ/fpqNwUao
U/3jp4nvc76K/FO14W56I/hXb23/0s8zzyny3gHfqcEu8Dn8OpNWDY4fZ4g9vQFB
hmv71HjJl0NRvvJHrXYmCEwlWPQjzru+8lj4JhBx/9ChKskTpVB6vkV//IX5Od1O
Zvaxh5x+xPCSKEgbmjv0uxaXtvnBJQa4xdMM7eHglGDSbZ2A13cg1qtxrCn05f6N
Bc4EiyOT2iofDDtqoxdLZPb4L6UDIR+EY1o+1l1mDBrBvqn6hQtpUoi+bgWe+xtS
ry30qmJkjjkejkJKJk+258uUI622kjlCCVGijj2145x9vnXXINiuOIuIj1K/a2dj
kP+2A3Jvt53z8gv9Jij9xC90725pCl5Cziw4XsBsg+jJJEn4IpqvwgoA/7SkDpZp

/ZSoVRgMfDvn60mzc/0Y6dtaX4FTsyJiduQBtKNtssGSVQGajOKEcfUOVgslkwuX
IPIODGoHEdFC4feve5uuucMbHw8pmjI0dYGz0XIcU5dZNW1yVvNaPXC7cKvIeuKS
F3bogXenDzZ40/6+n9kRRS74vzdOMv5CSoxQOrQw0pBvWm0DyUFRTJ53GZAfbEz+
1IU+cwAMmQR7FMpbJtaKJeNdccHe/nOm4kdnW6W00FxUVeUvbmcuRL8wVMHvXo58
6qDuHOk0LPXK+KLRr5P1QyD7b78t4PJOmbKgT0xQd8h1Oun2j61ZfQsvaguF0dM+
QOO+EWoUU0+1I4eCzMG38R927w9kT8jJCPmIF2DT5tSB0JWIMC+Md6u0HFKUPG2C

qbSB58Ykljvoiu70Avay79vAAREvjkjlVWYKLJMjiuvaweRGPWtKdeBXwOHNSFRY
1JekLYeGaSX0WzVcxQcA3flpGL+4SdjdRWDYK3wXv6QoQ9YVag78nMIYUECtz+Yt

py8dTIjdp3d+KDsJ8t0dYkvHETiv8QoDNeutIZZXgP0PhR1smfcEFeUTwe56nDDp
BJJsyaybQhj76+tz1346gymRTEasBTlklnmu6XafYJ290fklsfdjkYjklaqoviDZ
1OphMGkNCqUa0JslpPBuPbVAgEBB4R3MUNQZpR9W7GlIMW8KNBNtbn6qFYaMq2uG
6AmwTZAVfhru0yjnIELj3k3t/OS/YbA6wRFpg0GddNNRAgMBAAE=

`pragma protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip",
data_method="aes128-cbc"

`pragma protect encoding=(enctype="base64", line_length=76, bytes=128)

`pragma protect data_block
RgKC7i4hx7zh3MLd50RYrZoCwPWFEyLwISIXDLkpkL6qFgFm1WmZEwFvZjNfQCNUgoSHeIRpxg9i
lXnvMiBjQCiQVvMp32UtfSX625K8+yvJLMPdHQ8G/2qxa6ViHAhBhRcsSUl0XGskRmU3JvNuNfAk
0IoB1HpFEJ0Vv6vEI5g=

`pragma protect end_protected endmodule
```

2. Securing Your IP Core (Ask a Question)

As an IP vendor, you must protect your Intellectual Property and package your IP core in such a way that it is interoperable with EDA tools without compromising security. Encryption is managed at two levels (see Figure 2-1):

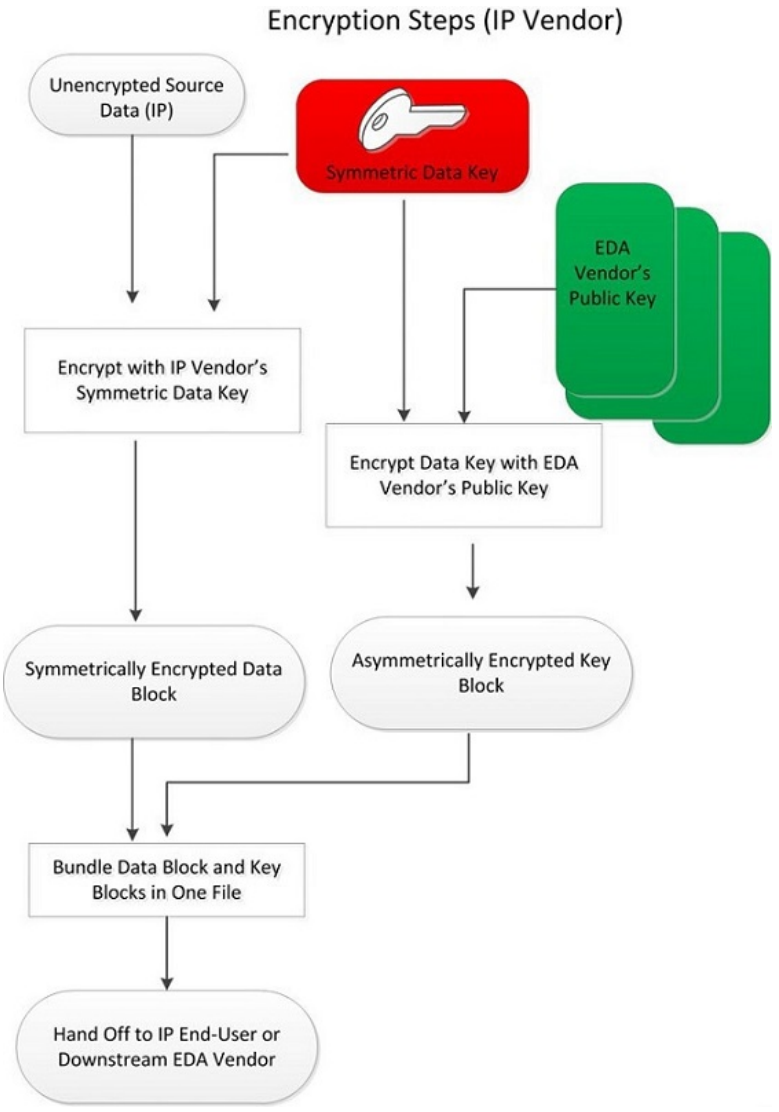
- IP Core encryption
- Data Key encryption

2.1. Encryption of IP Core with IEEE 1735-2014 Scheme (Ask a Question)

Perform the following steps to encrypt the IP core with IEEE 1735-2014 scheme.

1. Obtain the Public Key (see section Public Key from EDA Vendors) from each downstream EDA tool vendor.
2. Add the Encryption Envelopes to the RTL code (see section Encryption Envelopes). Ensure that all required EDA tool vendors are included.
3. Execute the encryptP1735 Perl script.

Figure 2-1. IP Encryption



2.2. Public Key from EDA Vendors (Ask a Question)

Obtain a public key from every downstream EDA tool vendor.

Important: Starting with Libero version 2023.2, the public key file (shown below) that contains all three public keys along with the Perl script to encrypt your files that are installed with the software at the following locations:

```
# IEEE1735v1 script and keys

<libero installation folder>/Designer/data/vault/ieee1735v1/
encrypt_ieee1735v1.pl
<libero installation folder>/Designer/data/vault/ieee1735v1/public_keys.txt
```

For Libero versions earlier than version 2023.2, click this link to download the public key file containing all three public keys along with the Perl script to encrypt your files. For any issues, file a technical support case at: microchip.my.site.com/s/newcase.

2.3. Adding an Encryption Envelope to Your RTL (Ask a Question)

You must add the Encryption envelopes (see section Encryption Envelopes) to the RTL codes. All EDA tools that need access to the encrypted data block must be included and identified as a key owner in the Encryption envelope.

Following is an example of a Verilog IP core and a VHDL IP core with an Encryption envelope. The envelope identifies Microchip, Synopsys, and Siemens as key owners.

2.3.1. Verilog IP Core with Encryption Envelope (Ask a Question)

The following shows an example of a Verilog IP core with an encryption envelope.

```
module secret (a, b, sum, clk, rstn); input[7:0]a, b;
input clk, rstn; output[8:0]sum; reg[8:0]sum;
`pragma protect version=1
`pragma protect encoding=(enctype="base64")
`pragma protect author="author-a", author_info="author-a-details"
`pragma protect encrypt_agent="encryptP1735.pl", encrypt_agent_info="Synplify encryption
scripts"
`pragma protect key_keyowner="Synplicity",key_keyname="SYNP05_001",key_method="rsa",key_block
`pragma protect key_keyowner="Mentor Graphics Corporation",key_keyname="MGC-VERIF- SIM-
RSA-1",key_method="rsa",key_block
`pragma protect key_keyowner="Microsemi Corporation",key_keyname="MSC-IP-KEY-
RSA",key_method="rsa",key_block
`pragma protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip", data_method="aes128-cbc"
`pragma protect begin
always @(posedge clk or negedge rstn) begin if (!rstn)
sum <= 9'b0; else
sum <= a + b; end
`pragma protect end endmodule
```

2.3.2. VHDL IP Core with Encryption Envelope (Ask a Question)

The following shows an example of a VHDL IP core with an encryption envelope.

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
generic(n: natural :=2); port(clock:in std_logic; clear:in std_logic; count:in std_logic;
Q:out std_logic_vector(n-1 downto 0)
);
end counter;
architecture behv of counter is
signal Pre_Q: std_logic_vector(n-1 downto 0); begin
`pragma protect version=1
`pragma protect encoding=(enctype="base64")
```

```
`pragma protect author="author-a", author_info="author-a-details"
`pragma protect encrypt_agent="encryptP1735.pl", encrypt_agent_info="Synplify encryption
scripts"
`pragma protect key_keyowner="Synplicity",key_keyname="SYNP05_001",key_method="rsa",key_block
`pragma protect key_keyowner="Mentor Graphics Corporation",key_keyname="MGC-VERIF- SIM-
RSA-1",key_method="rsa",key_block
`pragma protect key_keyowner="Microsemi Corporation",key_keyname="MSC-IP-KEY-
RSA",key_method="rsa",key_block
`pragma protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip", data_method="aes128-cbc"
`pragma protect begin
process(clock, count, clear) begin
if clear = '1' then
Pre_Q <= Pre_Q - Pre_Q;
elsif (clock='1' and clock'event) then if count = '1' then
Pre_Q <= Pre_Q + 1; end if;
end if;
end process;
Q <= Pre_Q;
`pragma protect end end behv;
```

2.3.3. Pragma Keywords (Ask a Question)

The following table lists the Pragma keywords in the Encryption envelope.

Table 2-1. Pragma Keywords

Pragma Keywords	Description
begin	Opens a new encryption envelope
end	Closes an encryption envelope
begin_protected	Opens a new decryption envelope
end_protected	Closes a decryption envelope
author	Identifies the author of an envelope
author_info	Specifies additional author information
encoding	Specifies the coding scheme for the encrypted data
data_keyowner	Identifies the owner of the data encryption key
data_method	Identifies the data encryption algorithm
data_keyname	Specifies the name of the data encryption key
data_public_key	Specifies the public key for data encryption
data_decrypt_key	Specifies the data session key
key_keyowner	Identifies the owner of the key encryption key
key_method	Specifies the key encryption algorithm
key_keyname	Specifies the name of the key encryption key
key_public_key	Specifies the public key for key encryption
key_block	Begins an encoded block of key data
version	P1735 encryption version

2.4. encryptP1735.pl Script (Ask a Question)

Execute the `encryptP1735.pl` script to encrypt your IP. The `encryptP1735` script is a Perl script that Synopsys provides to IP vendors for encryption of their IP cores.

Notes:

- Before running the script, make sure that the OpenSSL is installed on your machine. OpenSSL is required for the script to work.
- For Windows OS, it is recommended that the script is executed in the Cygwin Environment on Windows.

The following example command invokes the script with a random key to encrypt the data block:

```
perl./encryptP1735.pl -input secret.v -output secret_enc.v -pk
public_keys.txt -v -om encrypted
```

where:

Table 2-2.

Command Parameters	Description
-input *secret.v*	Specifies `secret.v` as the input file to the script. The input file is the non-encrypted HDL file containing one or more encryption envelopes.
-output *secret_enc.v*	Specifies `secret_enc.v` as the name of the encrypted output file after running the encryption script.
-pk *public_keys.txt*	Specifies `public_keys.txt` as the public keys repository file. This file contains public keys for all downstream EDA tools. The public keys file must include public keys for all EDA vendors mentioned in the Encryption envelope.
-om *encrypted*	Specifies how the IP is treated when generating the synthesis netlist; encrypted is the default mode. In this mode, the same data key used for encryption of the IP is used in the output synthesis netlist.
-v	Specifies that the script runs in Verbose mode.

2.4.1. Encrypted Output Files (Ask a Question)

The output file generated by the script contains Pragma directives for decrypting the encrypted data (IP core) and the data key that encrypts the data. The following sections show examples of encrypted Verilog and VHDL output.

Example of Encrypted Verilog Output

```
module secret (a, b, sum, clk, rstn); input[7:0]a, b;

input clk, rstn; output[8:0]sum; reg[8:0]sum

`pragma protect begin_protected

`pragma protect version=1

`pragma protect author="author-a", author_info="author-a-details"

`pragma protect encrypt_agent="encryptP1735.pl", encrypt_agent_info="Synplify
encryption scripts"

`pragma protect key_keyowner="Synplicity", key_keyname="SYNP05_001",
key_method="rsa"

`pragma protect encoding=(enctype="base64", line_length=76, bytes=256)
```

Synopsis Key Block

```
`pragma protect key_block
NfR8W3gmxwh3Bj4QxA+Qi+BhD1CTnQv7KO4UGOOS27KzF4jtejZxAewyFaShFSqRn9tRNx+u7Ivw
1m2BydGyW7MAQx2ePgbrKQbRLaN8XF/iiUFUX0QXnWDZrxtgcVHULOsPXpwd25wNyeWQkTekAsln
ubKiFDfNySxaP5W3SboZE0pMLqH+mpZlcvKljlE30uOAQQLjECEBGj1KxMZQ2hhUKLrXz34+9p68
tVzbM/u1TbsXvdPcN23UItAxNPSH5ND75rAviq7ACIVawH87/m2RshSDSVcmz7ndMpSJRQOFe2pd
usuHdCFJm1YaEaCZYfqReV7RjCzbV48d3LPtoA==

`pragma protect key_keyowner="Mentor Graphics Corporation", key_keyname="MGC-
VERIF-SIM- RSA-1", key_method="rsa"
```

```
`pragma protect encoding=(enctype="base64", line_length=76, bytes=128)
```

Siemens Key Block

```
`pragma protect key_block boN+vsIsOJ/
Ihy7BF0MM2ZdaeYl2zoepUP9xdDVnlME3q5lgqZtPjMtPqTQDvwbree7NngmOUGVm WbggEEW/
UWYWajwld641fsggKfu7kcFcMhLLBu0WHUVFvQjRhdiqcBWbEKM39O0SCYTJnhQFPs0B
RZgdCwOPvZ4IEAUqx4U=

`pragma protect key_keyowner="Microsemi Corporation", key_keyname="MSC-IP-
KEY-RSA", key_method="rsa"

`pragma protect encoding=(enctype="base64", line_length=76, bytes=960)
```

Microchip Key Block

```
`pragma protect key_block
MIID4jANBgkqhkiG9w0BAQEFAAOCA88AMIIDygKCA8EAxvOR7+3o0rtdoggobQ7e
3LQ5Bhjfcudafujkinm+213ui89cvxjkaYKRDadsklgfklDGTFyiYUIKasKv3MrW
xbaIlfktti2lBBdU/SDV83mLYKzAqe20/SaZR5FAZH8cyuUPxYOviHQ/fpqNwUao
U/3jp4nvc76K/FO14W56I/hXb23/0s8zzyny3gHfqcEu8Dn8OpNWDY4fZ4g9vQFB
hmv71HjJl0NRvvJHrXYmCEwlWPQjzru+8lj4JhBx/9ChKskTpVB6vkV//IX5Od1O
Zvaxh5x+xPCSKEgbmjv0uxaXtvnBJQa4xdMM7eHglGDSbZ2A13cg1qtxrCn05f6N
Bc4EiyOT2iofDDtqoxdLZPb4L6UDIR+EY1o+1l1mDBrBvqn6hQtpUoi+bgWe+xtS
ry30qmJkjjkejkJKJk+258uUI622kjlCCVGijj2145x9vnXXINiuOIuIj1K/a2dj
kP+2A3Jvt53z8gv9Jij9xC90725pCl5Cziw4XsBsg+jJJEn4IpqvwgoA/7SkDpZp

/ZSoVRgMfDvn60mzc/0Y6dtaX4FTsyJiduQBtKNtssGSVQGajOKEcfUOVgslkwuX
IPIODGoHEdFC4feve5uuucMbHw8pmjI0dYGz0XIcU5dZNW1yVvNaPXC7cKvIeuKS
F3bogXenDzZ40/6+n9kRRS74vzdOMv5CSoxQOrQw0pBvWm0DyUFRTJ53GZAfbEz+
1IU+cwAMmQR7FMpbJtaKJeNdccHe/nOm4kdnW6W00FxUVeUvbmcuRL8wVMHvXo58
6qDuHOk0LPXK+KLRr5P1QyD7b78t4PJOmbKgT0xQd8h1Oun2j61ZfQsvaguF0dM+
QOO+EWoUU0+1I4eCzMG38R927w9kT8jJCPmIF2DT5tSB0JWIMC+Md6u0HFKUPG2C

qbSB58Ykljvoiu70Avay79vAAREvjkjlVWYKLJMjiuvaweRGPWtKdeBXwOHNSFRY
1JekLYeGaSX0WzVcxQcA3flpGL+4SdjdRWDYK3wXv6QoQ9YVag78nMIYUECtz+Yt
py8dTIjdp3d+KDsJ8t0dYkvHETiv8QoDNeutIZZXgP0PhR1smfcEFeUTwe56nDDp
BJJsyaybQhj76+tz1346gymRTEasBTlklnmu6XafYJ290fklsfdjkYjklaqoviDZ
1OphMGkNCqUa0JslpPBuPbVAgEBB4R3MUNQZpR9W7GlIMW8KNBNtbn6qFYaMq2uG
6AmwTZAVfhru0yjnIELj3k3t/OS/YbA6wRFpg0GddNNRAgMBAAE=

`pragma protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip",
data_method="aes128-cbc"

`pragma protect encoding=(enctype="base64", line_length=76, bytes=128)
```

Data (IP Core) Block

```
`pragma protect data_block

RgKC7i4hx7zh3MLd50RYrZoCwPWFEyLwISIXDLkpkL6qFgFm1WmZEwFvZjNfQCNUgoSHeIRpxg9i

lXnvMiBjQCiQVvMp32UtfSX625K8+yvJLMPdHQ8G/2qxa6ViHAhBhRcsSUl0XGskRmU3JvNuNfAk
0IoB1HpFEJ0Vv6vEI5g=

`pragma protect end_protected endmodule
```

Example of Encrypted VHDL Output

```
library ieee ;

use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;
```

```
entity counter is

generic(n: natural :=2); port(clock:in std_logic; clear:in std_logic;
count:in std_logic;

Q:out std_logic_vector(n-1 downto 0)

);

end counter;

architecture behv of counter is

signal Pre_Q: std_logic_vector(n-1 downto 0); begin

`protect begin_protected

`protect version=1

`protect author="author-a", author_info="author-a-details"

`protect encrypt_agent="encryptP1735.pl", encrypt_agent_info="Synplify
encryption scripts"

`protect key_keyowner="Synplicity", key_keyname="SYNP05_001",
key_method="rsa"

`protect encoding=(enctype="base64", line_length=76, bytes=256)
```

Synopsys Key Block

```
`protect key_block

EzupxwpLZCgcCoy7O42J4O6TjEXDsFHlEXYIfYKVXIsm/8incqBuPuWZ26osQcaegOtanunB7lPo
sTFjlZBLLgsDLE/Pl7j8PhcxhySoKy/8TkZClQf7osKMbfeFAMFtIOAqjGT4Ab2F9DdosbC6QkNY
FCVLJSk5nNBeA6bslznTicV416exZcHTV5tJycz2vkFVlRY+BBtcXlhBrxCZSguf9OwHkr0OcufC
jKaHE//kfF1dlJ1jjcuidCnJ5rOtG3BDWFQ7f/ClH6H9IkqikEfDy2qGO4Kz1N8OF6sH2MKCj4O5
ye7d1aH+QH3FrTmoNgnVg9f7McoZ0Ito4Z1qCQ==

`protect key_keyowner="Mentor Graphics Corporation", key_keyname="MGC-VERIF-
SIM-RSA-1", key_method="rsa"

`protect encoding=(enctype="base64", line_length=76, bytes=128)
```

Mentor Graphics Key Block

```
`protect key_block

Pfy8Cgmz1tqEDSqqkQ+/HYByVzO7Iq9WSlfEgti2EYSXVTU974UChUeOJwTJUA5z24gL1gI2QF3I
SYQs6NgHG84V+DMh9s3biK9UDHz4KJqa5Xrsx6QwvD6co3rZ09bzNPL8w9uGaPK40DXWTQbY0T6W
pDdIw9u4pvhII/2L5eY=

`protect key_keyowner="Microsemi Corporation", key_keyname="MSC-IP-KEY-RSA",
key_method="rsa"

`protect encoding=(enctype="base64", line_length=76, bytes=960)
```

Microchip Key Block

```
`protect key_block
MIID4jANBgkqhkiG9w0BAQEFAAOCA88AMIIDygKCA8EAxvOR7+3o0rtdoggobQ7e
3LQ5Bhjfcudafujkinm+213ui89cvxjkaYKRDadsklgfklDGTFyiYUIKasKv3MrW
xbaIlfktti2lBBdU/SDV83mLYKzAqe20/SaZR5FAZH8cyuUPxYOviHQ/fpqNwUao
U/3jp4nvc76K/FO14W56I/hXb23/0s8zzyny3gHfqcEu8Dn8OpNWDY4fZ4g9vQFB
hmv71HjJl0NRvvJHrXYmCEwlWPQjzru+8lj4JhBx/9ChKskTpVB6vkV//IX5Od1O
```

```
Zvaxh5x+xPCSKEgbmjv0uxaXtvnBJQa4xdMM7eHglGDSbZ2A13cg1qtxrCn05f6N
Bc4EiyOT2iofDDtqoxdLZPb4L6UDIR+EY1o+1l1mDBrBvqn6hQtpUoi+bgWe+xtS
ry30qmJkjjkejkJKJk+258uUI622kjlCCVGijj2145x9vnXXINiuOIuIj1K/a2dj
kP+2A3Jvt53z8gv9Jij9xC90725pCl5Cziw4XsBsg+jJJEn4IpqvwgoA/7SkDpZp

/ZSoVRgMfDvn60mzc/0Y6dtaX4FTsyJiduQBtKNtssGSVQGajOKEcfUOVgslkwuX
IPIODGoHEdFC4feve5uuucMbHw8pmjI0dYGz0XIcU5dZNW1yVvNaPXC7cKvIeuKS
F3bogXenDzZ40/6+n9kRRS74vzdOMv5CSoxQOrQw0pBvWm0DyUFRTJ53GZAfbEz+
1IU+cwAMmQR7FMpbJtaKJeNdccHe/nOm4kdnW6W00FxUVeUvbmcuRL8wVMHvXo58
6qDuHOk0LPXK+KLRr5P1QyD7b78t4PJOmbKgT0xQd8h1Oun2j61ZfQsvaguF0dM+
QOO+EWoUU0+1I4eCzMG38R927w9kT8jJCPmIF2DT5tSB0JWIMC+Md6u0HFKUPG2C

qbSB58Ykljvoiu70Avay79vAAREvjkjlVWYKLJMjiuvaweRGPWtKdeBXwOHNSFRY
1JekLYeGaSX0WzVcxQcA3flpGL+4SdjdRWDYK3wXv6QoQ9YVag78nMIYUECtz+Yt
py8dTIjdp3d+KDsJ8t0dYkvHETiv8QoDNeutIZZXgP0PhR1smfcEFeUTwe56nDDp
BJJsyaybQhj76+tz1346gymRTEasBTlklnmu6XafYJ290fklsfdjkYjklaqoviDZ
1OphMGkNCqUa0JslpPBuPbVAgEBB4R3MUNQZpR9W7GlIMW8KNBNtbn6qFYaMq2uG
6AmwTZAVfhru0yjnIELj3k3t/OS/YbA6wRFpg0GddNNRAgMBAAE=

`protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip",
data_method="aes128-cbc"

`protect encoding=(enctype="base64", line_length=76, bytes=288)
```

Data (IP core) Block

```
`protect data_block

+m/P6uHpXWo/2MDE8lnrIGmBHe6DSUtiNm7PkpwC+dMErJ9rG4vuwDcoqErHHk4oToYBn4ZavftY
DJc1W3U7+dxEN3lVcgRsWveZZ0ePIfkkEKhp7cSgfFt5kFfwPEoMHPDhAPeElMr84o0pYEiFdO6V
GwOJgULvGsFedDKwWnTn6O9FbtKBKuKyl8NG27C89GRtkr4UhguNgVDJKs/O8E9bHlSlyxSh2sD4
GnTPLAVC4NONi4HjsBhxVGvq04yjbJwOHohjI/WeY26ZqHJN7jqkKrdOhXTi/DRoCY15vjfvALr1
kzErv8zjc9qGqBWucHhmUgwfKzp6p8XfFPHTZlOnsKigVN9Q8Kmu6ZmN3nYadlK8ASo4A7q3v9mA
otx6

`protect end_protected end behv;
```

2.5. Packaging and Bundling the Encrypted IP and the Data Key (Ask a Question)

When you execute the `encryptP1735.pl` script, you bundle and package the Encrypted IP and the Encrypted Data Key together in one single file, which is the output of the script. This file is ready for delivery to your customers.

3. Running Libero SoC with Encrypted IP (Ask a Question)

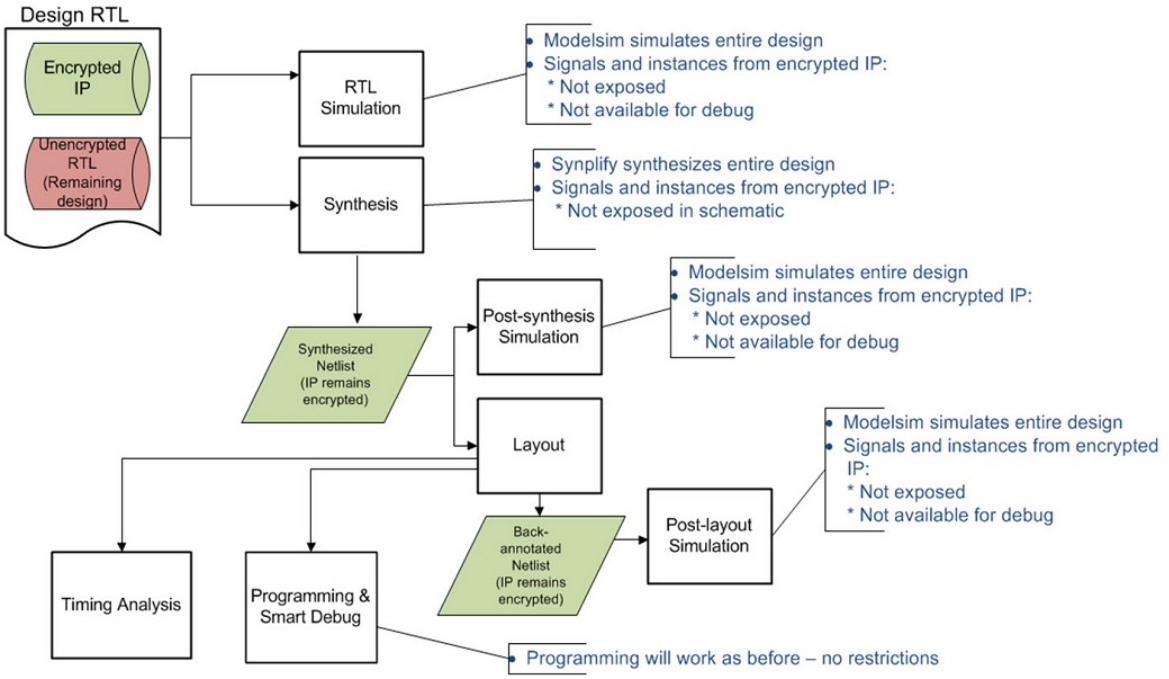
Libero SoC software v11.3 or later supports the use of third-party encrypted IP cores in the design flow for SmartFusion 2, IGLOO 2, RTG4, and PolarFire families (Figure 3-1).

3.1. To run Libero SoC with Encrypted IP (Ask a Question)

The Libero SoC software support for encrypted IP is enabled by default. Perform the following the steps to incorporate an encrypted IP inside the Libero software.

1. Set your Project Settings so that the synthesized output is a Verilog netlist.
2. Import the encrypted IP core as HDL (see section Import Encrypted IP Core as HDL).
3. Run synthesis (see section Run Synthesis) and simulation (see section Run ModelSim Simulation).
4. The following figure shows how to run the remaining Libero SoC design flow.

Figure 3-1. Encrypted IP Design Flow



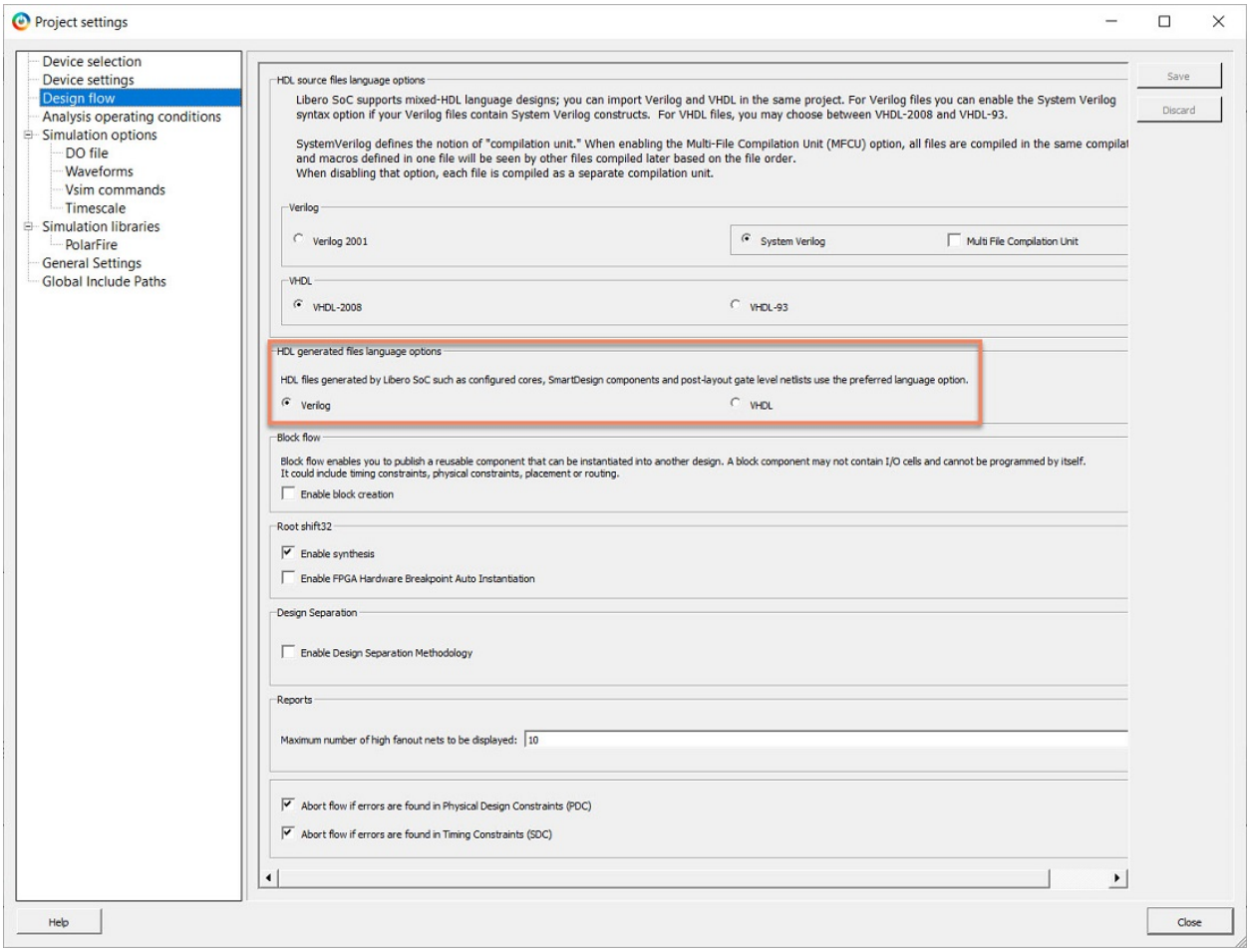
3.2. Encrypted IP Design Flow Must Use Verilog Netlist from Synthesis (Ask a Question)

When a new project is created, you must change Project Settings to support the IEEE 1735-2014 secure IP flow. You can then import the encrypted IP core as Verilog or VHDL source files.

The IEEE 1735-2014 scheme supports only Verilog as the netlist format; EDIF format is not supported. You must set Libero SoC Project Settings to use the Verilog netlist from Synthesis.

1. From the Project menu, choose **Project Settings > Design Flow**.
2. Select **Verilog** as the HDL generated file language option as shown in Figure 3-2.
3. Click **Save** and then **Close.**

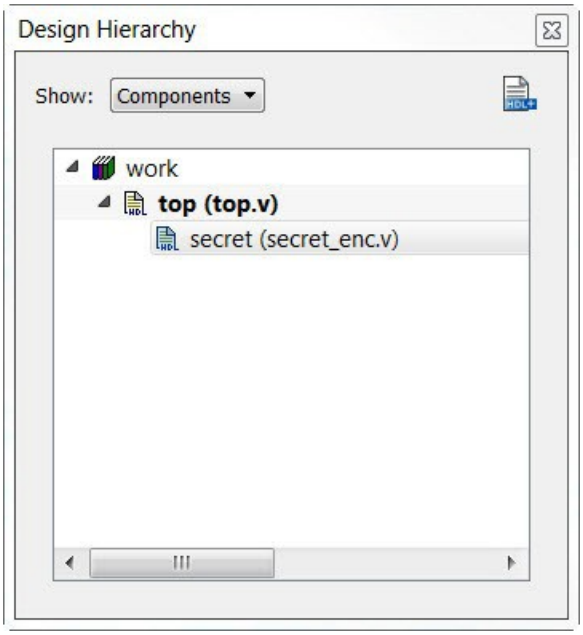
Figure 3-2. Project Settings



3.3. Import Encrypted IP Core as HDL (Ask a Question)

Import the encrypted IP HDL and the non-encrypted HDL file as HDL source files (**File > Import > HDL Source Files**). The Design Hierarchy window displays the imported file in your design.

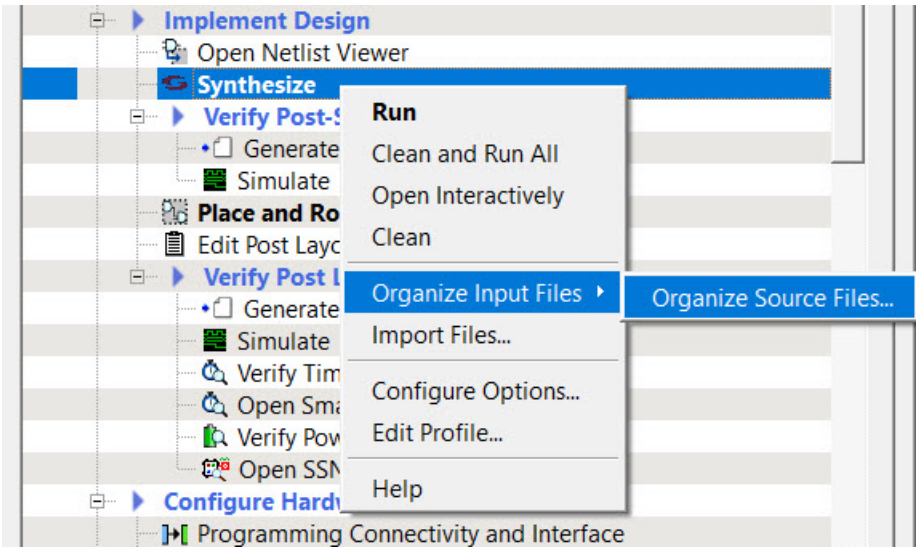
Figure 3-3. Design Hierarchy



Important: It is recommended that the encrypted IP be presented as a single file. If the IP is currently organized in a hierarchy of files, it is recommended that the entire IP be concatenated into a single file after encryption. Currently, if the encrypted IP is defined in multiple files, the user must pass the (lower level) files manually to synthesis and RTL simulation steps. This is done from **Organize input file** option Synthesis/Simulation tool, as shown in Figure 3-4. See the `Organize Source` file in *Libero Help* for more information on how to organize source files.

Figure 3-4 shows how to organize input source files for Synthesis.

Figure 3-4. Organizing Input Source Files for Synthesis



3.3.1. Smart Design Support (Ask a Question)

SmartDesign is a visual block-based design creation tool for instantiation, configuration, and connection of Microchip IP, user-generated IP, and the custom/glue-logic HDL modules. Encrypted IP can also be instantiated in a SmartDesign, along with another non-encrypted IP. See the *About SmartDesign* document in *Libero Help* for more information.

3.4. Run Synthesis (Ask a Question)

After synthesis, only the interface signals (inputs and output ports) of the Secure IP core are visible in the RTL and Technology views (Figure 3-5 and Figure 3-6). Signals and instance names that are internal to the Encrypted IP are not visible.

Figure 3-5. Synplify Pro RTL View

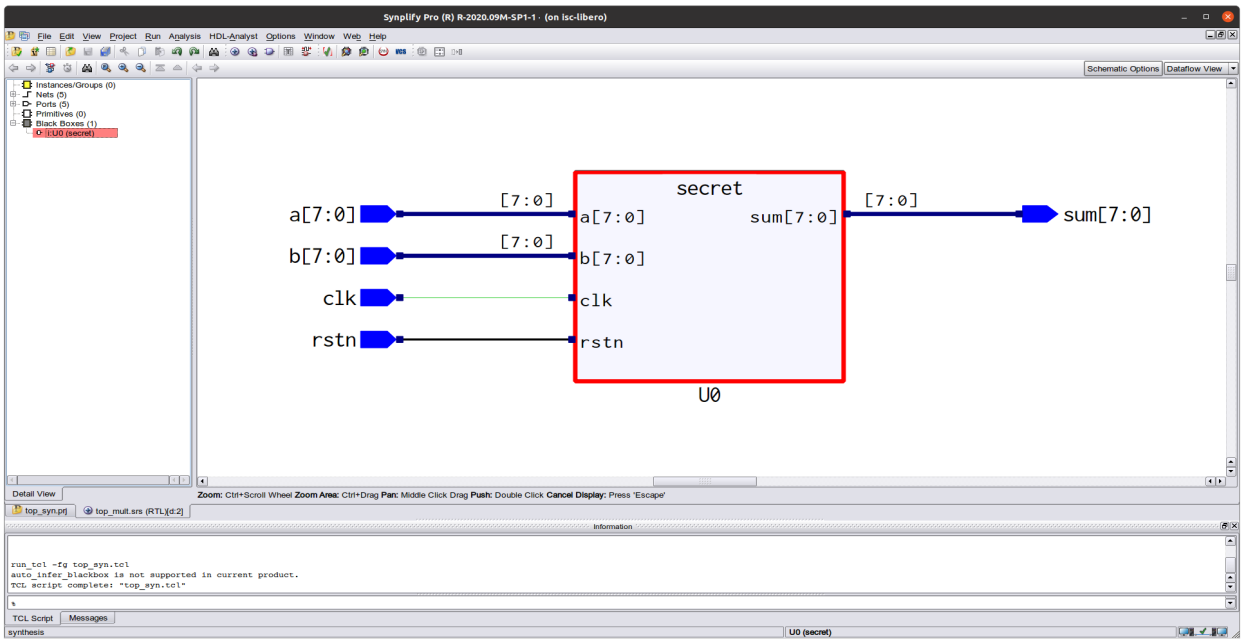
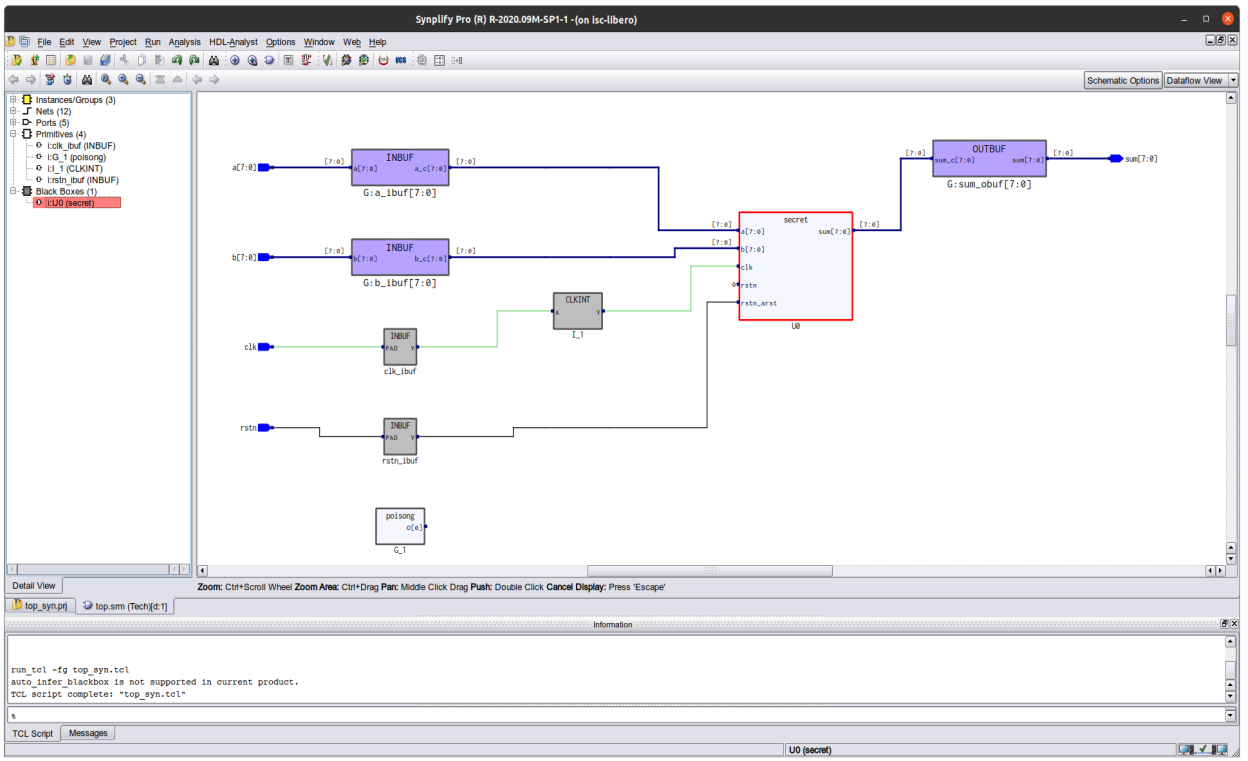


Figure 3-6. SynplifyPro Technology View



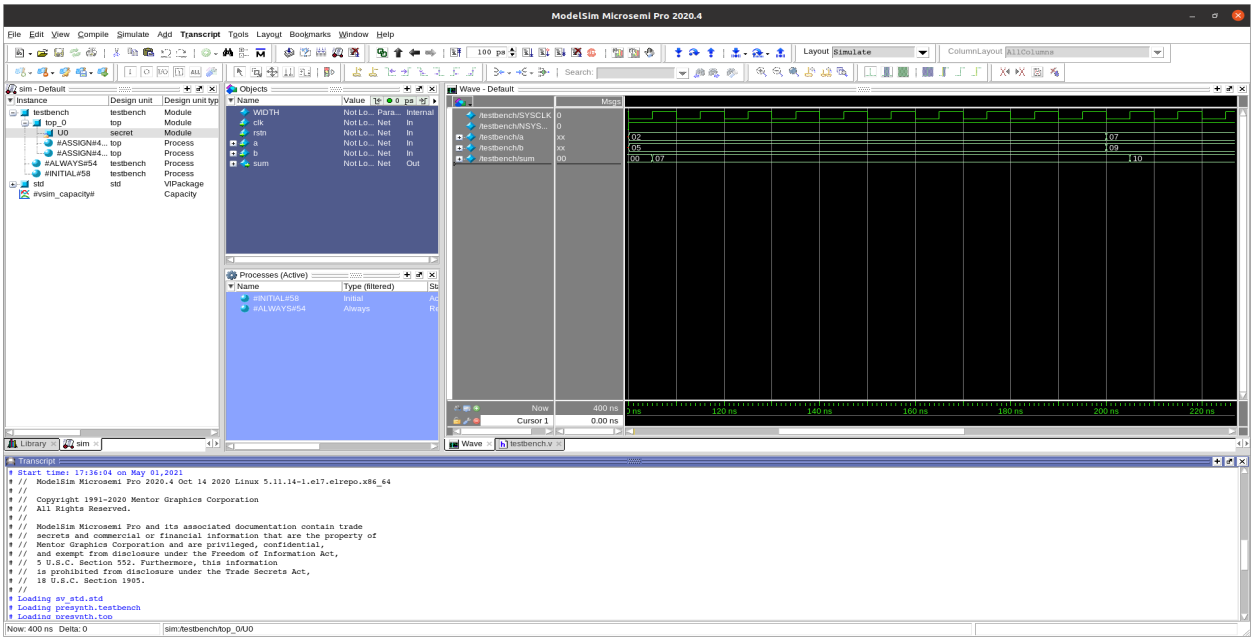
In the RTL and Technology Views, the Push and Pop commands are disabled for design blocks encrypted with the IEEE 1735-2014. You cannot push into the encrypted IP block ‘U0’ to look at the internal signals, nets, or instances inside the encrypted block.

3.5. Run ModelSim Simulation (Ask a Question)

ModelSim simulates the entire design for pre-synthesis, post-synthesis, and post-layout simulations. However, the signals and instances internal to the encrypted IP are not exposed and are not available for debug.

The values of the internal signals are not displayed in the waveform window; only the interface signals at the boundary of the encrypted IP instance ‘U0’ are displayed.

Figure 3-7. Modelsim Simulation of Encrypted IP Core



Note: Simulation is supported for both Verilog and VHDL.

3.6. Libero SoC and Encrypted IPs (Ask a Question)

Libero SoC Software processes designs with encrypted IP through the entire Design without compromising the encrypted IP content. The encryption of IP is protected in Synthesis and Simulation tools, as mentioned in previous sections.

All netlists exported from Libero SoC have the IP component encrypted. These include:

- Back annotated netlist after Place and Route: `*_ba.v` or `*_ba.vhd`
- Exported netlist after Compile: `*.v` or `*.vhd`

Microchip adheres to the Encryption Guidelines provided in the IEEE 1735-2014 standard throughout the design flow.

3.6.1. Example (Ask a Question)

This section shows an example in which an Encrypted module is implemented using the Libero SoC Secure IP flow.

The example consists of the following files:

- `Secret.v`: This is a simple Non-Encrypted Verilog module. This module has encryption envelopes, as shown in section Encryption Envelopes.
- `Secret_enc.v`: This is the encrypted version of Secret.v module that has been encrypted by executing the `encryptP1735.pl` script on the Secret.v module.
- `Top.v`: This is a top-level module instantiating encrypted secret_enc.v module. `Tb.v` is the test bench for the Top.v module.
- `Public_keys.txt`: This text file contains Public Keys from Synopsys, Siemens, and Microchip, as shown in section Public Key from EDA Vendors.

3.6.1.1. Encryption of IP Module (Ask a Question)

We are going to use to `encryptP1735.pl` script which implements IEEE 1735-2014 standard for encryption of IP modules (`secret.v` in this example). The segment of the code that needs to

be encrypted have to be included within Encryption envelopes. (see section Adding an Encryption Envelope to Your RTL).

All Public Keys from vendors supporting this standard are stored in a single file `Public_Keys.txt`. Execute `encryptP1735.pl` script with the `secret.v` as input file and `secret_enc.v` as output file.

The following figure shows an example of output after `encryptP1735.pl` has been executed on the secret.v module.

Figure 3-8. Output of EncryptP1735.pl Script

```
File  Edit  View  Search  Terminal  Help

$perl ./encryptP1735.pl -i secret.v -o secret_enc.v -pk public_keys.txt -om encrypted
Info: found openssl encryption engine in /usr/bin
Info: HDL type is set to Verilog.
Info: Found key 'Synplicity.SYNP05_001' in the repository
Info: Found key 'Mentor Graphics Corporation.MGC-VERIF-SIM-RSA-1' in the repository
Info: Found key 'Microsemi Corporation.MSC-IP-KEY-RSA' in the repository
Generating Synplicity encryption version 1 key-block
Info: using openssl for RSA 2048 bit encryption
Generating encryption version 1 key-block
Info: using openssl for RSA 2048 bit encryption
Generating encryption version 1 key-block
Info: using openssl for RSA 2048 bit encryption
Generating aes256-cbc encrypted data-block
Info: using openssl for AES256-CBC encryption
Info: Processed 1 envelopes
$
```

The output file is similar to the one shown in section Encrypted Output Files.

The encrypted output source file of the IP has key_blocks corresponding to all the vendors and Data_blocks with encrypted information.

Note: See section encryptP1735.pl Script for more information about different parameters of the script. The script can be executed on both Windows and Linux OS with OpenSSL and Perl Installed.

3.6.1.2. Importing Encrypted IP in Libero SoC (Ask a Question)

Perform the following steps to import an Encrypted module. The Encrypted module can be imported in the same way you import any HDL file into a Libero Project.

1. Create a Libero Project with SmartFusion 2/IGLOO 2/RTG4/PolarFire family die.
2. Import the files `Top.v` and `Secret_enc.v` files into the Libero Project using the following path: **File > import > HDL Source Files**.
3. Import the corresponding Test bench file `tb.v` into the Libero Project using the following path:**File > Import > HDL Stimulus Files**.
4. Upon importing these files, your design hierarchy and stimulus hierarchy appear, as shown in Figure 3-9 and Figure 3-10.

Figure 3-9. Design Hierarchy

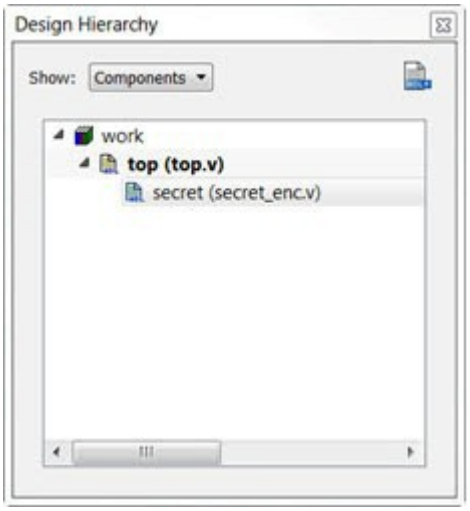
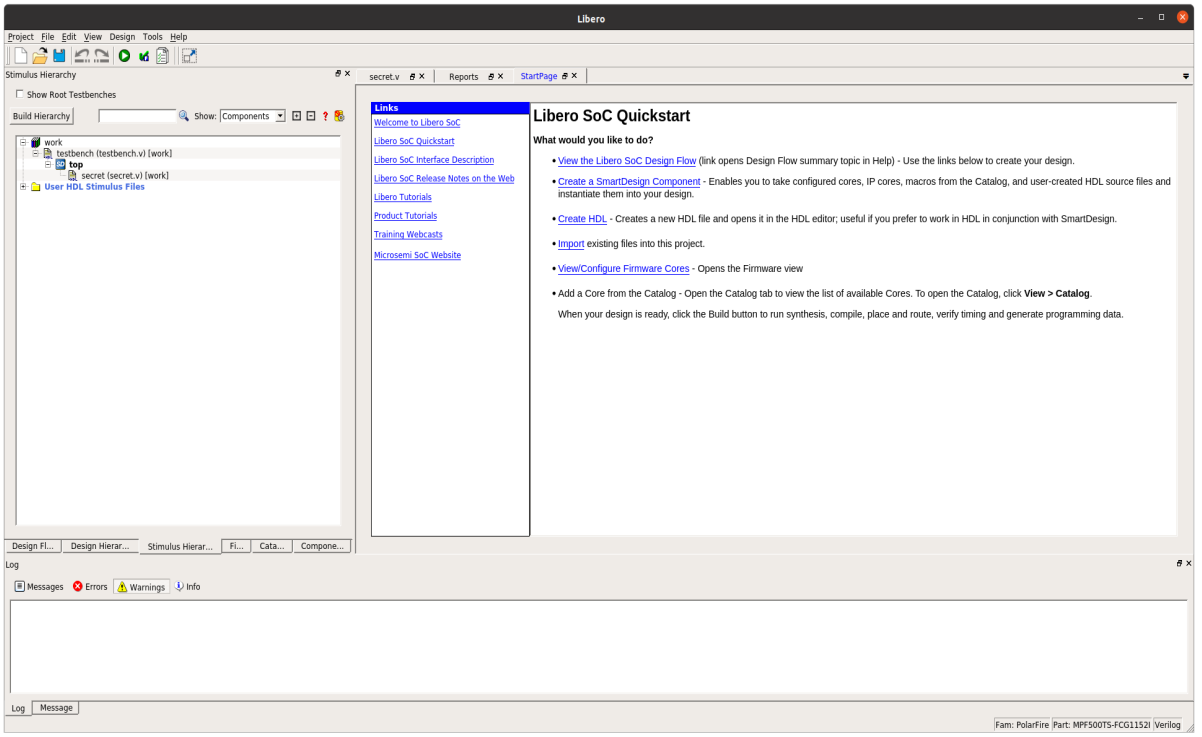
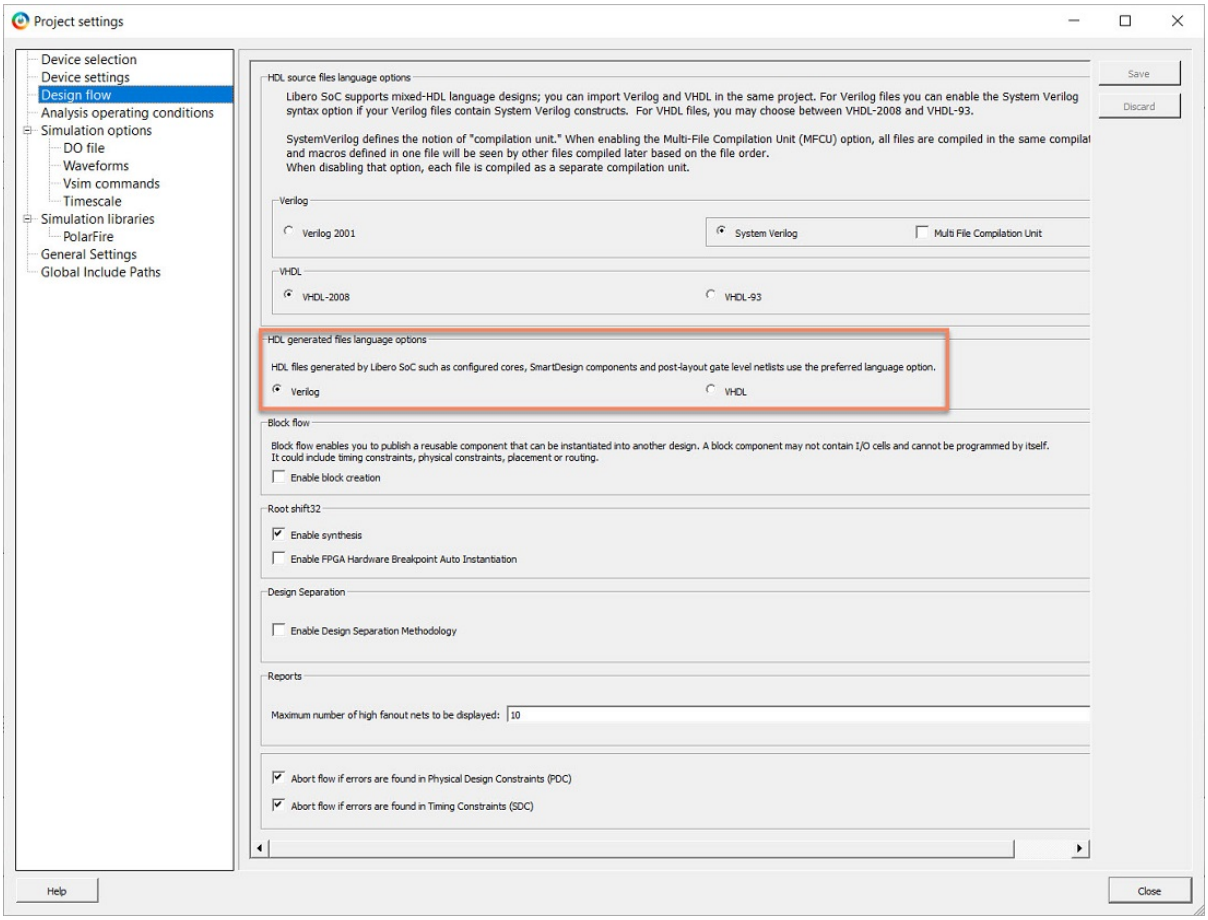


Figure 3-10. Stimulus Hierarchy



5. Within a Top Level module or Smart Design, it's possible to have multiple instantiations of an encrypted module. To designate top.v as the Root module, select **Set as Root**.
6. Change the HDL-generated file language option to Verilog by accessing the Libero Project Settings Menu through the following path: **Project > Project Settings > Design Flow**. For visual guidance, see Figure 3-11.

Figure 3-11. Project Settings for Netlist Format



3.6.1.3. Synthesis (Ask a Question)

The Synthesis tool (Synplify Pro) decrypts the protected content using Synopsys Key Block present in Encrypted module `secret_enc.v`. After synthesis, only the interface signals (inputs and output ports) of secure IP core are visible in the RTL and Technology views. See section Run Synthesis for more information. The Verilog netlist file (`.vm` file) obtained after synthesis does not show internal instances of encrypted module and this information is again re-encrypted by the Synthesis tool.

3.6.1.4. Simulations (Ask a Question)

The Simulation tool (ModelSim) decrypts the protected content using the ModelSim Key Block present in Encrypted module secret_enc.v. ModelSim simulates the entire design for pre-synthesis, post-synthesis, and post-layout simulations. However, the signals and instances internal to the encrypted IP are not exposed and are not available for debug. See section Run ModelSim Simulation for more information.

3.6.1.5. Compile and Layout (Ask a Question)

The rest of the tools in the Libero SoC Design Flow decrypt the protected content using Microchip Key Block present in Encrypted module secret_enc.v.

Once the synthesis is completed, the Compile tool takes the encrypted `.vm` netlist file as input for further processing by the Layout tool. The execution and output of these tools are similar to the Regular flow.

Note: Constraints flow, including Timing Constraints and Floorplan Constraints, are not supported for instances inside encrypted blocks. In the above example, Constraint flow is not supported for secret_enc.v module. However, you can provide constraints to the interface of the Encrypted module.

3.6.1.6. Generate Back Annotated Files (Ask a Question)

Once the Layout is complete, you can generate the Back Annotated Files for Post-Layout simulations. The `*_ba.v` or `*_ba.vhd` files generated show the internal information of secure_enc.v module as encrypted. These files incorporate Key_Block from Siemens, which is used for decryption while running Post-Layout simulations.

3.6.1.7. Generate Programming Data (Ask a Question)

Once the design has completed the Layout and Post-Layout simulations, you can generate the programming file.

4. Frequently Asked Questions (Ask a Question)

Following is a list of FAQs about Secure IP flow and its support in Libero SoC.

Are VHDL simulations supported, as we are using a Verilog Netlist?

Secure IP flow is supported for both VHDL and Verilog. Mixed mode simulation is not required if the design and test bench are both in VHDL. The Verilog netlist is only required for passing the design from the synthesis to compile step in Libero. Post-synthesis and other simulation steps still use VHDL netlist, if the preferred input HDL type is VHDL at Project Creation.

Is Microchip Block flow supported in Secure IP flow?

No. Block flow is not supported for Encrypt IP and Secure IP flow.

Are parameters/generics supported?

Yes. Secure IP flow works on an Encrypted IP with parameters or generic definitions. However, leaving top level parameters/generics and ports unencrypted makes the RTL easier to integrate.

See the VHDL example in this document, which has a generic definition.

Which versions of Perl and OpenSSL are required for `encryptP1735.pl` script?

Any version of OpenSSL/Perl can be used for the script to execute.

How is OpenSSL installed?

OpenSSL is Open-Source Software. Most Linux Installations have OpenSSL pre-installed.

For Windows, you must install OpenSSL.exe. You can download the application from the OpenSSL website. Once you install OpenSSL on Windows, you need to set the PATH environment variable to `<openssl_installation_dir>\bin` for the `EncryptP1735.pl` to work.

Can we import an encrypted Verilog core into a VHDL design, and vice versa?

Yes. You can import an Encrypted Verilog (or VHDL) module in a VHDL (or Verilog) Design.

5. Revision History (Ask a Question)

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision	Date	Description
G	05/2025	The following is the list of changes made in revision G of the document: • Updated the paths for Keys and script in section Public Key from EDA Vendors. • Updated section VHDL IP Core with Encryption Envelope.
F	11/2024	The following is the list of changes made in revision F of the document: • In section Public Key from EDA Vendors, provided a link for filing a technical support case and removed sample keys.
E	08/2024	This document is released with Libero SoC Design Suite v2024.2 without changes from v2024.1.
D	02/2024	The following is the list of changes made in revision D of the document: • In section Introduction, added PolarFire SoC to the list of design families. • In section Libero SoC Secure IP Design Flow Requirements, updated the descriptions for Hardware Requirements and Public Keys for encryption. • Incorporated general editing improvements.
C	02/2024	In section Public Key from EDA Vendors, revised the way in which users of Libero v2024.1 and later can obtain the public key file.
B	08/2023	In section Public Key from EDA Vendors, revised the way in which users of Libero v2023.2 and later can obtain the public key file.
A	05/2021	Initial Revision.

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 979-8-3371-1145-2

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.