The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN204427

Software Switch For Dual-Ethernet FM3 Microcontroller

Associated Part Family:	Series Name	Product Number
	MB9BF610 MB9BFB10	MB9BF616/617/618 MB9BFD16/17/18

High end FM3 derivates contain up to two independent Ethernet MAC (EMAC) modules. Both Ethernet interfaces have been designed to work completely independent to maximize flexibility and increase suitability for safety and fault-safe applications.

Contents

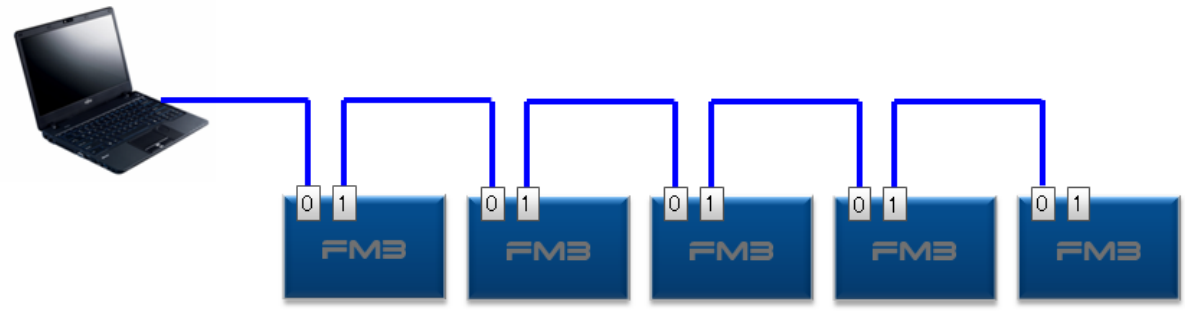
1 Preface

High end FM3 derivates contain up to two independent Ethernet MAC (EMAC) modules. Both Ethernet interfaces have been designed to work completely independent to maximize flexibility and increase suitability for safety and fault-safe applications.

However, there are scenarios where a chained topology is favored. Applications like lighting control and non real-time critical industrial automation benefit from reduced hardware costs, easier deployment because of simpler wiring effort and smaller physical size requirements.

In other words, instead of connecting several nodes through Ethernet switches, hubs or routers, they carry two interfaces each and are connected to each other via a single cable.

Figure 1. Daisy-chain topology



Cypress offers a software example of how such a software switch might be implemented on an dual-Ethernet FM3 microcontroller.

This application note explains the function and properties of this software example.

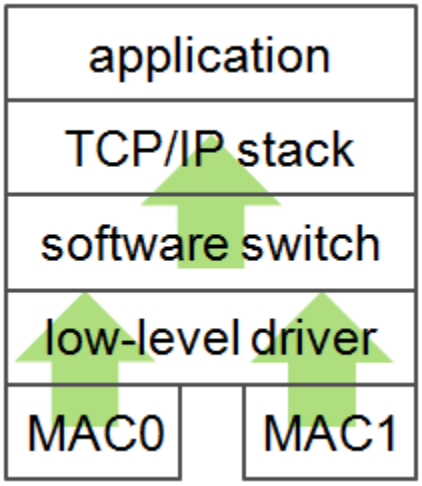
2 General Structure

This chapter explains how the software ethernet switch works

2.1 Structure

The Software Switch acts as an abstraction layer on top of the Cypress Ethernet low-level driver and provides a “virtual EMAC” API. Thus, you can use this API to send and receive data just like if you used the actual hardware EMAC units directly. The difference is that both underlying Ethernet interfaces can serve as a switch to enable daisy-chain topology.

Figure 2. Software Switch Layer



2.2 Function

The logic is rather simple:

Send all frames on both interfaces

If incoming frame is for me: pass to application

If incoming frame is broadcast: pass to application and other interface

If incoming frame is not for me: pass to other interface

3 Using the Ethernet Software Switch

How to set-up and use the ethernet software switch

3.1 General use

First of all, `Ethswitch_Init()` must be used to initialize the internal data structures and underlying hardware.

`Ethswitch_Autonegotiate()` must be called regularly to react to changed connection states like a plucked or reconnected cable.

`Ethswitch_RxFrame()` must be called regularly and as often as possible to ensure reception and fast switching function.

Do not use functions from emac.c/h directly when this software switch is running! E.g. use `Ethswitch_RxFrame()` instead of `Emac_RxFrame()`[1].

3.2 User Settings in emac_user.h

emac_user.h configures the underlying Ethernet low-level driver. It is recommended not to change the provided settings as they are necessary for the switch to work correctly.

That said there are two exceptions:

3.2.1 Ring Size

If you have to save RAM, you might experiment with different values for

EMAC0_TX_RING_SIZE,

EMAC0_RX_RING_SIZE,

EMAC1_TX_RING_SIZE and

EMAC1_RX_RING_SIZE.

However, if those settings are chosen to small, the switch will lose frames if too high network traffic occurs.

3.2.2 MAC Address

The MAC address has to be identical for both EMAC units. This is ensured by defining `EMAC1_MAC_ADDRESSx` to `EMAC0_MAC_ADDRESSx`. Therefore, to change the device's MAC address, you have just to alter the six definitions from `EMAC0_MAC_ADDRESS0` to `EMAC0_MAC_ADDRESS5`.

However, there is one exception you might consider for optimizing your network load: It is safe to use Emac_TxFrame() to send frames through only one specific physical interface instead of Ethswitch_TxFrame(), which utilizes both.

Switch API

3.2.3 Ethswitch_Init()

This function initializes the Ethernet Software Switch, including both physical EMAC instances.

Prototype	
en_result_t Ethswitch_Init(void)	
Return Values	Description
Ok	Switch and underlying hardware initialized

3.2.4 Ethswitch_DeInit()

This function de-initializes both underlying EMAC instances.

Prototype	
en_result_t Ethswitch_DeInit(void)	
Return Values	Description
Ok	Switch and underlying hardware deinitialized

3.2.5 Ethswitch_TxFrame()

Transmit a data frame via Ethernet Software Switch.

This function will transmit data through both underlying Ethernet interfaces.

Prototype	
en_result_t Ethswitch_TxFrame(uint8_t * pau8Buf, uint16_t u16Len)	
Parameter Name	Description
[in] pu8buf	Address of buffer containing data to be sent
[in] u16len	Number of bytes to be sent
Return Values	Description
Ok	Channel disabled
ErrorOperationInProgress	No free Tx buffers at one or two channels

3.2.6 Ethswitch_GetFrameLength()

Check how many bytes are in Ethernet Software Switch's rx buffer without taking them out. This functions returns the length of any received frame, regardless of the destination. If more than 0 bytes have been received, Ethswitch_RxFrame() should be called soon.

This function can be used to determine how much memory has to be allocated if dynamic memory management is used.

Prototype	
uint16_t Ethswitch_GetFrameLength(void)	
Return Values	Description
length	Number of bytes received, 0 if nothing received

3.2.7 Ethswitch_RxFrame()

This function contains the switching logic. For short latency, call it as often as possible!

This function will receive and copy data addressed for this node to the buffer addressed by pu8Buf.

If `Ethswitch_GetFrameLength()` was used before, this function will use the correct underlying EMAC instance.

This function returns the number of bytes only if the received frame was addressed at this node, i.e. it will return 0 if it is neither a broadcast nor meant for this node's address.

Prototype	
uint16_t Ethswitch_RxFrame (uint8_t * pau8Buf)	
Parameter Name	Description
[in] pau8Buf	Buffer address to store the data
Return Values	Description
length	Number of received bytes.

3.2.8 Ethswitch_GetLinkStatus()

Read out link status flags from underlying PHYs to determine if link is up or down

Prototype	
en_emac_link_status_t Ethswitch_GetLinkStatus(void)	
Return Values	Description
EMAC_LinkStatusLinkUp	if one or both PHY links are up
EMAC_LinkStatusLinkDown	if both PHY links are down

3.2.9 Ethswitch_Autonegotiate()

Trigger autonegotiation for both underlying Ethernet interfaces

This function configures link parameters (speed, duplex) automatically.

If return value is EMAC_LinkStatusLinkUp, transmission and reception is enabled,

so Ethswitch_TxFrame() and Ethswitch_RxFrame() can be used.

Before this function, Ethswitch_Init() must be called.

It is recommended to call this function regularly to react on disconnected and reconnected cables.

Prototype	
en_emac_link_status_t Ethswitch_Autonegotiate(void)	
Return Values	Description
EMAC_LinkStatusLinkUp	if one or both PHY links are up
EMAC_LinkStatusLinkDown	if both PHY links are down
EMAC_LinkStatusAutonegotiationInProgress EMAC_LinkStatusAutonegotiationSuccessful EMAC_LinkStatusAutonegotiationNotSupported EMAC_LinkStatusInvalidParameter EMAC_LinkStatusUnknownError	if condition occurs at one or both underlying EMACs. The upper two usually do not persist for more than one function call. The latter indicate a systematic problem, which must be fixed outside switch software.

3.3 Usage Examples

This section explains the usage of the FM3 Ethernet switch by concrete example.

3.3.1 How to set up an Ethernet interface

As the hardware settings are done by the Ethernet switch abstraction layer, setup is simple:

```
// Board pins setting for using Ethernet
ConfigureEthernetPins();

// callback functions setting for EMAC0
stcEmacConfig.pfnRxCallback = EMAC0RxCallbackFunc;
stcEmacConfig.pfnTxCallback = EMAC0TxCallbackFunc;

// Initialize  Ethernet switch and establish link
Ethswitch_Init();
Ethswitch_Autonegotiate();
```

It is recommended to call `Ethswitch_Autonegotiate()` regularly to react to changed connection states.

3.3.2 How to Send a Frame

The user can transmit a frame by calling `Ethswitch_TxFrame()` after initialization. This function has two parameters; 1st the address of the buffer to be sent and 2nd the number of bytes to be sent by this function.

The buffer must contain the protocol headers and payload beforehand. The following example sets up and transmits a UDP frame.

```
uint8_t ua8EthBuf[1500];
uint8_t ua8Payload[] = "Konnichiwa World!”;

// creating an Ethernet frame with IP and UDP header
TxBufferUDPFill(ua8EthBuf, ua8Payload, sizeof(ua8Payload));
ua8EthBuf[42 + sizeof(ua8Payload) - 3] = (u8NumberOfCalls + '0');

// transmit the frame
Ethswitch_TxFrame(ua8EthBuf, 42 + sizeof(ua8Payload));
```

(The function TxBufferUDPFill() writes header information of an Ethernet, IP and UDP message into ua8EthBuf and concatenates contents of ua8Payload. It is only used to produce a valid Ethernet frame to transmit via Ethswitch_TxFrame(). Normally, this task would be done by a TCP/IP stack.)

3.3.3 How to Receive a Frame

Reception is done simply by calling `Ethswitch_RxFrame()` like this:

```
p->len =  Ethswitch_RxFrame(p->payload);
```

As already mentioned before in this document, `Ethswitch_RxFrame()` should be called as often as possible to maximize network throughput as it contains the switching logic.

4 Performance Measurement

This chapter explains how latency and bandwidth has been measured and presents the results of those measurements with.

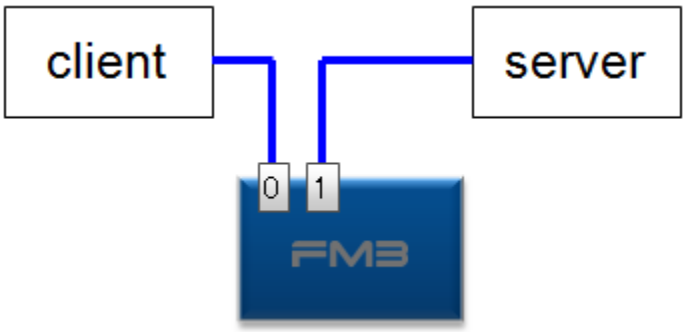
4.1 Preface

The actual performance depends on the software architecture, CPU load and concrete system configuration. As a frame is forwarded by calling Ethswitch_RxFrame(), the frequency of this call has certainly the highest impact on performance figures.

This chapter explains the measurement methods and results for the software example mb9bfxxx_ethernet_switch_uip-v11.

As the direction from EMAC0 towards EMAC1 is prioritized, all measurements have been conducted both ways.

Figure 3. Setup for performance tests



For all measurements mentioned in this document, software example *mb9bfxxx_ethernet_switch_uip,* version 1.1 was used.

4.2 Latency

Latency is the propagation delay between a frame being received by an interface and forwarded by the other.

Figure 4. Measurement setup

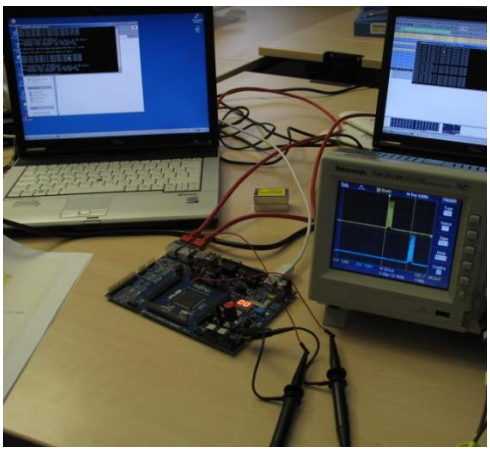
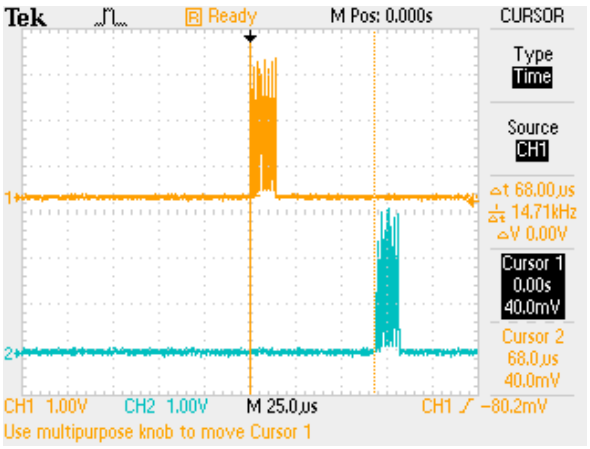


Figure 5. Latency of 128 byte payload



The test setup is like in Figure 3 but with a 2-channel oscilloscope connected to the RX0 signal of EMAC0 and TX0 line of EMAC1 (and vice versa for the second set of data).

To measure the latency according to the different frame sizes, the ping command with the –l parameter is called.

On the host, designated as server in above picture, wireshark is running to ascertain that truly the ICMP packets were regarded and not an ARP or other protocol.

Packet size	Latency MAC0 → MAC1 [usec]	Latency MAC1 → MAC0 [usec]
1 (46)	25	24
64	39	37
128	58	56
256	94	96
512	170	172
786	252	256
1024	324	320
1472	456	460

Compared to a typical hardware switch, the Cypress FM3 software switch's latency is approximately four times as high.

4.3 Bandwidth

Bandwidth is the time that is necessary to transmit a given amount of information. To measure it, there can be set-up a server of some kind and count the time necessary to download the data.

There can be used a webserver with a large file, which is downloaded with a webserver or a program like wget[2] or curl[3], which yield statistics.

As bandwidth might depend largely on frame size, another test method has been applied. Two dedicated test programs have been used with a server and a client part that send test data with a defined packet size. The chosen software is NetIO[4] and iperf[5]

However, bandwidth measurement figures have shown to be very variable, in fact test results differed about 30% from each other. This is caused by the use of an ordinary, non-deterministic Windows host and comparatively short measurement periods.

However, this effect is not an error but expected behavior in real-world applications.

[2] Available at http://sourceforge.net/projects/wget/

[3] Available at http://sourceforge.net/projects/curl/

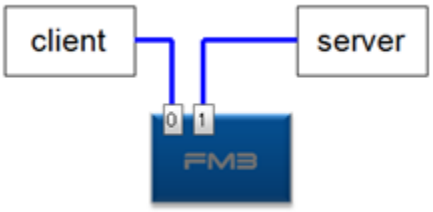
[4] Available at http://www.ars.de/ars/ars.nsf/docs/netio

[5] Available at http://iperf.sourceforge.net/

4.3.1 NetIO

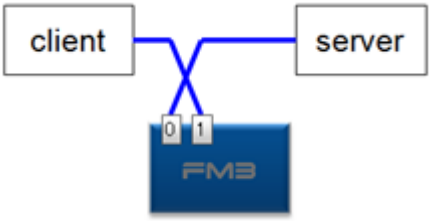
On server: linux-i386 –s On client: win32-i386.exe –t 192.168.1.50

Transfer via Software Switch from EMAC0 to EMAC1



Packet size	Tx rate [KByte/sec]	Rx rate [KByte/sec]
1k	2324.46	4967.9
2k	3334.13	4552.97
4k	3922.50	4649.94
8k	3841.08	4822.37
16k	3990.49	4555.05
32k	3882.90	3328.26

Transfer via Software Switch from EMAC1 to EMAC0



Packet size	Tx rate [KByte/sec]	Rx rate [KByte/sec]
1k	844.13	3405.85
2k	1484.97	4090.07
4k	2895.54	3858.86
8k	2642.54	3605.11
16k	3619.33	3604.33
32k	4400.98	3778.18

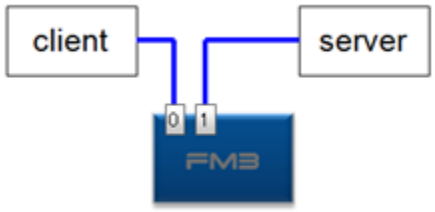
Transfer via direct cable (for comparison)

Packet size	Tx rate [KByte/sec]	Rx rate [KByte/sec]
1k	9380.18	8528.35
2k	9318.68	8386.59
4k	9535.79	8268.96
8k	9529.09	8542.06
16k	9539.51	8467.90
32k	9649.60	8511.25

4.3.2 iperf

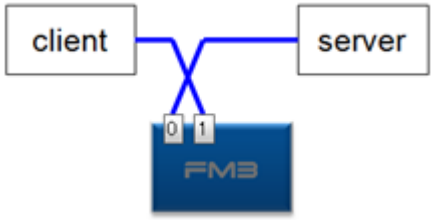
On server: iperf –s On client: iperf –c 192.168.1.50 –f KBytes –d

Transfer via Software Switch from EMAC0 to EMAC1




```
C:\>iperf -c 192.168.1.50 -f KBytes -d
------------------------------------------------------------
Client connecting to 192.168.1.50, TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.1 port 4528 connected with 192.168.1.50 port 5001
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
Waiting for server threads to complete. Interrupt again to force quit.
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.4 sec  27288 KBytes  2634 KBytes/sec
```

Transfer via Software Switch from EMAC1 to EMAC0




```
C:\>iperf -c 192.168.1.50 -f KBytes -d
------------------------------------------------------------
Client connecting to 192.168.1.50, TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.1 port 1114 connected with 192.168.1.50 port 5001
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
Waiting for server threads to complete. Interrupt again to force quit.
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  38896 KBytes  3890 KBytes/sec
```

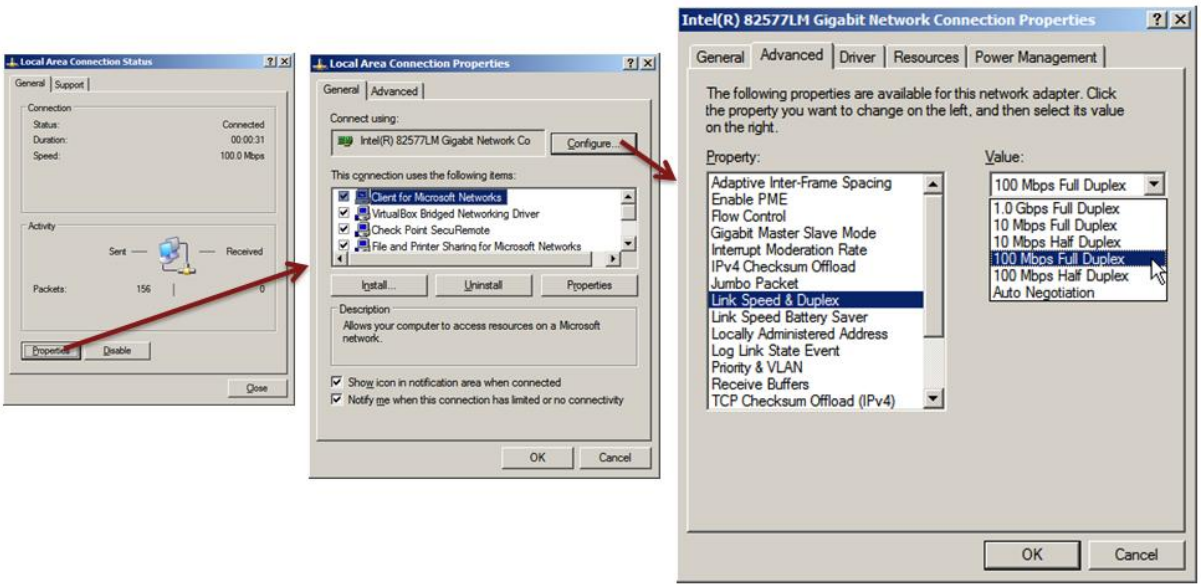
Transfer via direct cable (for comparison)


```
C:\>iperf -c 192.168.1.50 -f KBytes -d
------------------------------------------------------------
Client connecting to 192.168.1.50, TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.1 port 1114 connected with 192.168.1.50 port 5001
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
Waiting for server threads to complete. Interrupt again to force quit.
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  38896 KBytes  3890 KBytes/sec
```

Setting up the test PC for comparison measurements

As the FM3 supports IEEE802.3u Fast Ethernet with 100Mbit/sec but not Gigabit Ethernet like in most current PCs, the network interface card has to be set to 100Mbps mode if a a direct connection shall be compared with a connection via the Software Switch.

Figure 6. Setting Windows Network Speed Settings



5 Additional Information

Information about CYPRESS Semiconductor's Microcontroller can be found on the following Internet page:

http://www.cypress.com/cypress-microcontrollers

The software examples related to this application note are:

mb9bfxxx_ethernet_switch_uip

mb9bfxxx_ethernet_ switch_lwip

All FM3 examples can be found on the following Internet page:

http://www.cypress.com/cypress-mcu-softwareexamples

6 Document History

Document Title: AN204427 - Software Switch for Dual-Ethernet FM3 Microcontroller

Document Number: 002-04427

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	_	CHNO	03/12/2013	Initial Release
*A	5034296	CHNO	01/19/2015	Converted Spansion Application Note “AN706-00066-1v0-E” to Cypress format
*B	5885152	AESATMP9	09/15/2017	Updated logo and copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709