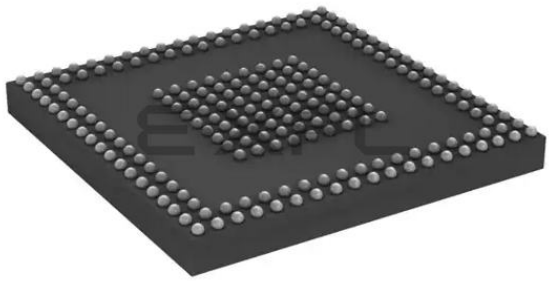

Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	XCore
Core Size	32-Bit 12-Core
Speed	1000MIPS
Connectivity	Configurable
Peripherals	-
Number of I/O	73
Program Memory Size	128KB (32K x 32)
Program Memory Type	SRAM
EEPROM Size	-
RAM Size	-
Voltage - Supply (Vcc/Vdd)	3V ~ 3.6V
Data Converters	A/D 8x12b
Oscillator Type	Internal
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Surface Mount
Package / Case	217-LFBGA
Supplier Device Package	217-FBGA (16x16)
Purchase URL	https://www.e-xfl.com/product-detail/xmos/xs1-u12a-128-fb217-c10

- **Channels and channel ends** Tasks running on logical cores communicate using channels formed between two channel ends. Data can be passed synchronously or asynchronously between the channel ends assigned to the communicating tasks. Section 7.5
- **xCONNECT Switch and Links** Between tiles, channel communications are implemented over a high performance network of xCONNECT Links and routed through a hardware xCONNECT Switch. Section 7.6
- **Ports** The I/O pins are connected to the processing cores by Hardware Response ports. The port logic can drive its pins high and low, or it can sample the value on its pins optionally waiting for a particular condition. Section 7.3
- **Clock blocks** xCORE devices include a set of programmable clock blocks that can be used to govern the rate at which ports execute. Section 7.4
- **Memory** Each xCORE Tile integrates a bank of SRAM for instructions and data, and a block of one-time programmable (OTP) memory that can be configured for system wide security features. Section 10
- **PLL** The PLL is used to create a high-speed processor clock given a low speed external oscillator. Section 8
- **USB** The USB PHY provides High-Speed and Full-Speed, device, host, and on-the-go functionality. Data is communicated through ports on the digital node. A library is provided to implement USB device functionality. Section 11
- **JTAG** The JTAG module can be used for loading programs, boundary scan testing, in-circuit source-level debugging and programming the OTP memory. Section 15

1.1 Software

Devices are programmed using C, C++ or xC (C with multicore extensions). XMOS provides tested and proven software libraries, which allow you to quickly add interface and processor functionality such as USB, Ethernet, PWM, graphics driver, and audio EQ to your applications.

1.2 xTIMEcomposer Studio

The xTIMEcomposer Studio development environment provides all the tools you need to write and debug your programs, profile your application, and write images into flash memory or OTP memory on the device. Because xCORE devices operate deterministically, they can be simulated like hardware within xTIMEcomposer: uniquely in the embedded world, xTIMEcomposer Studio therefore includes a static timing analyzer, cycle-accurate simulator, and high-speed in-circuit instrumentation.

xTIMEcomposer can be driven from either a graphical development environment, or the command line. The tools are supported on Windows, Linux and MacOS X and available at no cost from xmos.com/downloads. Information on using the tools is provided in the xTIMEcomposer User Guide, X3766.

7.2 xTIME scheduler

The xTIME scheduler handles the events generated by xCORE Tile resources, such as channel ends, timers and I/O pins. It ensures that all events are serviced and synchronized, without the need for an RTOS. Events that occur at the I/O pins are handled by the Hardware-Response ports and fed directly to the appropriate xCORE Tile. An xCORE Tile can also choose to wait for a specified time to elapse, or for data to become available on a channel.

Tasks do not need to be prioritised as each of them runs on their own logical xCORE. It is possible to share a set of low priority tasks on a single core using cooperative multitasking.

7.3 Hardware Response Ports

Hardware Response ports connect an xCORE tile to one or more physical pins and as such define the interface between hardware attached to the XS1-U12A-128-FB217, and the software running on it. A combination of 1bit, 4bit, 8bit, 16bit and 32bit ports are available. All pins of a port provide either output or input. Signals in different directions cannot be mapped onto the same port.

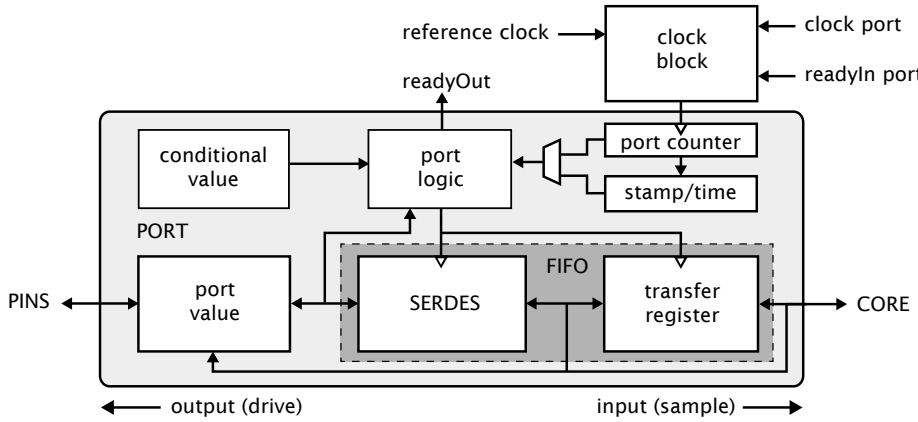

Figure 5: Port block diagram

The port logic can drive its pins high or low, or it can sample the value on its pins, optionally waiting for a particular condition. Ports are accessed using dedicated instructions that are executed in a single processor cycle.

Data is transferred between the pins and core using a FIFO that comprises a SERDES and transfer register, providing options for serialization and buffered data.

Each port has a 16-bit counter that can be used to control the time at which data is transferred between the port value and transfer register. The counter values can be obtained at any time to find out when data was obtained, or used to delay I/O until some time in the future. The port counter value is automatically saved as a timestamp, that can be used to provide precise control of response times.

Feature	Bit	Description
Disable JTAG	0	The JTAG interface is disabled, making it impossible for the tile state or memory content to be accessed via the JTAG interface.
Disable Link access	1	Other tiles are forbidden access to the processor state via the system switch. Disabling both JTAG and Link access transforms an xCORE Tile into a “secure island” with other tiles free for non-secure user application code.
Secure Boot	5	The processor is forced to boot from address 0 of the OTP, allowing the processor boot ROM to be bypassed (*see* §9).
Redundant rows	7	Enables redundant rows in OTP.
Sector Lock 0	8	Disable programming of OTP sector 0.
Sector Lock 1	9	Disable programming of OTP sector 1.
Sector Lock 2	10	Disable programming of OTP sector 2.
Sector Lock 3	11	Disable programming of OTP sector 3.
OTP Master Lock	12	Disable OTP programming completely: disables updates to all sectors and security register.
Disable JTAG-OTP	13	Disable all (read & write) access from the JTAG interface to this OTP.
Disable Global Debug	14	Disables access to the DEBUG_N pin.
	21..15	General purpose software accessable security register available to end-users.
	31..22	General purpose user programmable JTAG UserID code extension.

Figure 12: Security register features

port with resource ID 0x200100, and the OTP control is on a 16-bit port with ID 0x100300. Programming is performed through `libotp` and `xburn`.

10.2 SRAM

Each xCORE Tile integrates a single 64KB SRAM bank for both instructions and data. All internal memory is 32 bits wide, and instructions are either 16-bit or 32-bit. Byte (8-bit), half-word (16-bit) or word (32-bit) accesses are supported and are executed within one tile clock cycle. There is no dedicated external memory interface, although data memory can be expanded through appropriate use of the ports.

10.3 Deep Sleep Memory

The XS1-U12A-128-FB217 device includes 128 bytes of deep sleep memory for state storage during sleep mode. Deep sleep memory is volatile and if device input power is remove, the data will be lost.

15 JTAG

The JTAG module can be used for loading programs, boundary scan testing, in-circuit source-level debugging and programming the OTP memory.

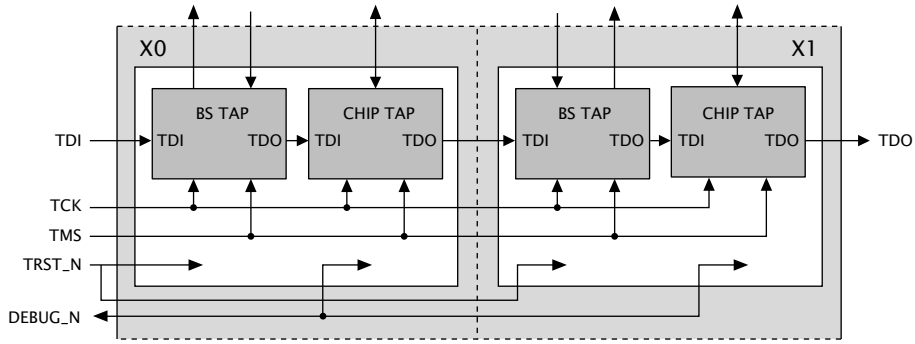

Figure 16: JTAG chain structure

The JTAG chain structure is illustrated in Figure 16. Directly after reset, three TAP controllers are present in the JTAG chain for each xCORE Tile: the debug TAP, the boundary scan TAP and the processor TAP. The debug TAP provides access into the peripherals including the ADC and USB. The boundary scan TAP is a standard 1149.1 compliant TAP that can be used for boundary scan of the I/O pins. The processor TAP provides access into the xCORE Tile, switch and OTP for loading code and debugging.

The JTAG module can be reset by holding TMS high for five clock cycles.

The DEBUG_N pin is used to synchronize the debugging of multiple processors. This pin can operate in both output and input mode. In output mode and when configured to do so, DEBUG_N is driven low by the device when the processor hits a debug break point. Prior to this point the pin will be tri-stated. In input mode and when configured to do so, driving this pin low will put the processor into debug mode. Software can set the behavior of the processor based on this pin. This pin should have an external pull up of 4K7-47KΩ or left not connected in single core applications.

The JTAG device identification register can be read by using the IDCODE instruction. Its contents are specified in Figure 17.

Figure 17: IDCODE return value

Bit31				Device Identification Register																								Bit0
Version				Part Number																Manufacturer Identity								1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	0	1	1	0	0	1	1
0				0				0				0				3				6				3				3			

- Avoid corners in the trace. Where necessary, rather than turning through a 90 degree angle, use two 45 degree turns or an arc.
- DO NOT route USB traces near clock sources, clocked circuits or magnetic devices.
- Avoid stubs on high speed USB signals.

16.3 Land patterns and solder stencils

The land pattern recommendations in this document are based on a RoHS compliant process and derived, where possible, from the nominal *Generic Requirements for Surface Mount Design and Land Pattern Standards* IPC-7351B specifications. This standard aims to achieve desired targets of heel, toe and side fillets for solder-joints.

Solder paste and ground via recommendations are based on our engineering and development kit board production. They have been found to work and optimized as appropriate to achieve a high yield. These factors should be taken into account during design and manufacturing of the PCB.

The following land patterns and solder paste contains recommendations. Final land pattern and solder paste decisions are the responsibility of the customer. These should be tuned during manufacture to suit the manufacturing process.

The package is a 217 pin Fine Ball Grid Array package on a 0.8mm pitch with 0.4mm balls.

An example land pattern is shown in Figure 22.

Pad widths and spacings are such that solder mask can still be applied between the pads using standard design rules. This is highly recommended to reduce solder shorts.

16.4 Ground and Thermal Vias

Vias next to every other ground ball into the ground plane of the PCB are recommended for a low inductance ground connection and good thermal performance. Vias with with a 0.6mm diameter annular ring and a 0.3mm drill would be suitable.

16.5 Moisture Sensitivity

XMOS devices are, like all semiconductor devices, susceptible to moisture absorption. When removed from the sealed packaging, the devices slowly absorb moisture from the surrounding environment. If the level of moisture present in the device is too high during reflow, damage can occur due to the increased internal vapour pressure of moisture. Example damage can include bond wire damage, die lifting, internal or external package cracks and/or delamination.

All XMOS devices are Moisture Sensitivity Level (MSL) 3 - devices have a shelf life of 168 hours between removal from the packaging and reflow, provided they

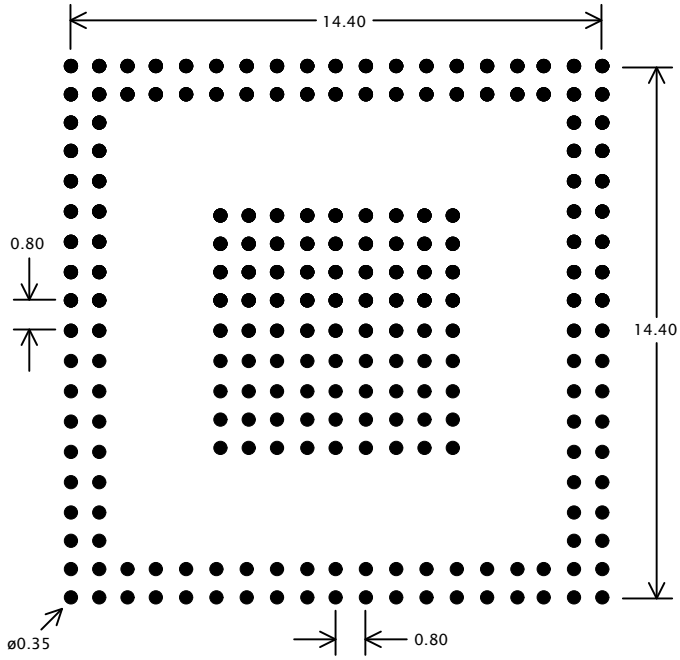

Figure 22: Example land pattern

are stored below 30C and 60% RH. If devices have exceeded these values or an included moisture indicator card shows excessive levels of moisture, then the parts should be baked as appropriate before use. This is based on information from *Joint IPC/JEDEC Standard For Moisture/Reflow Sensitivity Classification For Nonhermetic Solid State Surface-Mount Devices* J-STD-020 Revision D.

17 Example XS1-U12A-128-FB217 Board Designs

This section shows example schematics and layout for a 2-layer PCB.

- Figures 23 shows example schematics and layout. It uses a 24 MHz crystal for the clock, and an SPI flash for booting. The XS1-U12A-128-FB217 is powered directly from 5V. An optional ESD protection device is included to increase ESD protection from 2 to 15 kV.
- Figures 24 shows example schematics and layout for a design that uses an oscillator rather than a crystal. If required a 3V3 oscillator can be used (for example when sharing an oscillator with other parts of the design), but a resistor bridge must be included to reduce the XI/CLK input from 3V3 to 1V8.
- Figure 25 shows example schematics and layout for a design that does not use USB and that runs off the internal 20 MHz oscillator. The XS1-U12A-128-FB217 is powered directly from 3V3.

Flash, AVDD, RST, and JTAG connectivity are all optional. Flash can be removed if the processor boots from OTP. The AVDD decoupler and wiring can be removed if the ADC is not used. RST_N and all JTAG wiring can be removed if debugging is not required (see Appendix M)

Appendices

A Configuring the device

The device is configured through ten banks of registers, as shown in Figure 43.

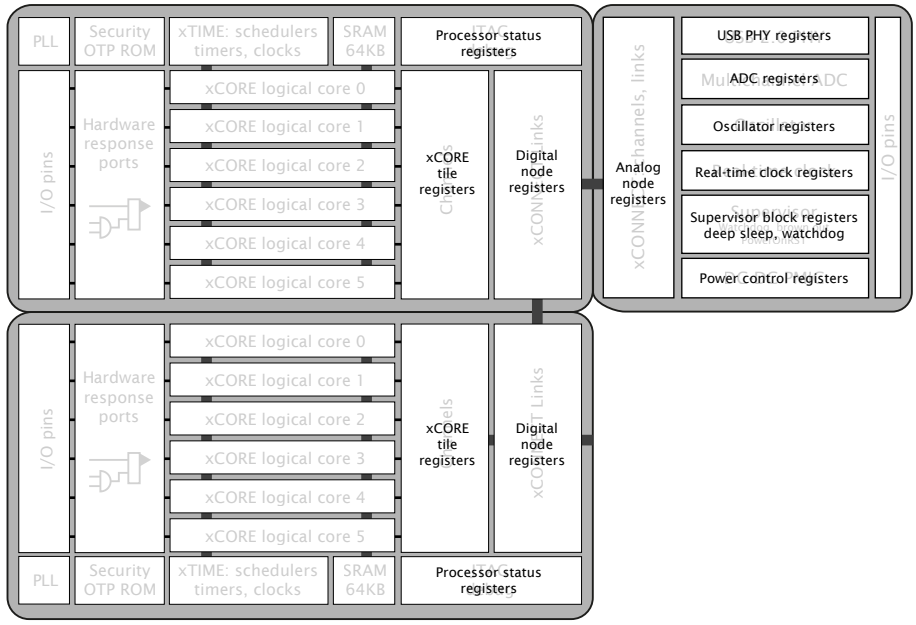

Figure 43: Registers

A.1 Accessing a processor status register

The processor status registers are accessed directly from the processor instruction set. The instructions GETPS and SETPS read and write a word. The register number should be translated into a processor-status resource identifier by shifting the register number left 8 places, and ORing it with 0x0C. Alternatively, the functions `getps(reg)` and `setps(reg,value)` can be used from XC.

A.2 Accessing an xCORE Tile configuration register

xCORE Tile configuration registers can be accessed through the interconnect using the functions `write_tile_config_reg(tileref, ...)` and `read_tile_config_reg(tile ↪ ref, ...)`, where `tileref` is the name of the xCORE Tile, e.g. `tile[1]`. These functions implement the protocols described below.

Instead of using the functions above, a channel-end can be allocated to communicate with the xCORE tile configuration registers. The destination of the channel-end should be set to `0xnnnnC20C` where `nnnnnn` is the tile-identifier.

A write message comprises the following:

control-token 192	24-bit response channel-end identifier	16-bit register number	32-bit data	control-token 1

The response to a write message comprises either control tokens 3 and 1 (for success), or control tokens 4 and 1 (for failure).

A read message comprises the following:

control-token 193	24-bit response channel-end identifier	16-bit register number	control-token 1

The response to the read message comprises either control token 3, 32-bit of data, and control-token 1 (for success), or control tokens 4 and 1 (for failure).

A.3 Accessing digital and analogue node configuration registers

Node configuration registers can be accessed through the interconnect using the functions `write_node_config_reg(device, ...)` and `read_node_config_reg(device, ...)`, where `device` is the name of the node. These functions implement the protocols described below.

Instead of using the functions above, a channel-end can be allocated to communicate with the node configuration registers. The destination of the channel-end should be set to `0xnnnnC30C` where `nnnn` is the node-identifier.

A write message comprises the following:

control-token 192	24-bit response channel-end identifier	16-bit register number	32-bit data	control-token 1

The response to a write message comprises either control tokens 3 and 1 (for success), or control tokens 4 and 1 (for failure).

A read message comprises the following:

control-token 193	24-bit response channel-end identifier	16-bit register number	control-token 1

The response to a read message comprises either control token 3, 32-bit of data, and control-token 1 (for success), or control tokens 4 and 1 (for failure).

A.4 Accessing a register of an analogue peripheral

Peripheral registers can be accessed through the interconnect using the functions `write_periph_32(device, peripheral, ...)`, `read_periph_32(device, peripheral, ...)`, `write_periph_8(device, peripheral, ...)`, and `read_periph_8(device, peripheral, ...)`; where device is the name of the analogue device, and peripheral is the number of the peripheral. These functions implement the protocols described below.

A channel-end should be allocated to communicate with the configuration registers. The destination of the channel-end should be set to 0xnnnnpp02 where nnnn is the node-identifier and pp is the peripheral identifier.

A write message comprises the following:

control-token 36	24-bit response channel-end identifier	8-bit register number	8-bit size	data	control-token 1

The response to a write message comprises either control tokens 3 and 1 (for success), or control tokens 4 and 1 (for failure).

A read message comprises the following:

control-token 37	24-bit response channel-end identifier	8-bit register number	8-bit size	control-token 1

The response to the read message comprises either control token 3, data, and control-token 1 (for success), or control tokens 4 and 1 (for failure).

0x04: Control PSwitch permissions to debug registers

Bits	Perm	Init	Description
31:1	RO	-	Reserved
0	CRW		Set to 1 to restrict PSwitch access to all CRW marked registers to become read-only rather than read-write.

C.5 Cause debug interrupts: 0x05

This register can be used to raise a debug interrupt in this xCORE tile.

0x05: Cause debug interrupts

Bits	Perm	Init	Description
31:2	RO	-	Reserved
1	RO	0	Set to 1 when the processor is in debug mode.
0	CRW	0	Set to 1 to request a debug interrupt on the processor.

C.6 xCORE Tile clock divider: 0x06

This register contains the value used to divide the PLL clock to create the xCORE tile clock. The divider is enabled under control of the tile control register

0x06: xCORE Tile clock divider

Bits	Perm	Init	Description
31:8	RO	-	Reserved
7:0	RW		Value of the clock divider minus one.

C.7 Security configuration: 0x07

Copy of the security register as read from OTP.

0x07: Security configuration

Bits	Perm	Init	Description
31:0	RO		Value.

C.8 PLink status: 0x10 .. 0x13

Status of each of the four processor links; connecting the xCORE tile to the switch.

0x10 .. 0x13: PLink status

Bits	Perm	Init	Description
31:26	RO	-	Reserved
25:24	RO		00 - ChannelEnd, 01 - ERROR, 10 - PSCTL, 11 - Idle.
23:16	RO		Based on SRC_TARGET_TYPE value, it represents channelEnd ID or Idle status.
15:6	RO	-	Reserved
5:4	RO		Two-bit network identifier
3	RO	-	Reserved
2	RO		1 when the current packet is considered junk and will be thrown away.
1	RO	0	Set to 1 if the switch is routing data into the link, and if a route exists from another link.
0	RO	0	Set to 1 if the link is routing data into the switch, and if a route is created to another link on the switch.

C.9 Debug scratch: 0x20 .. 0x27

A set of registers used by the debug ROM to communicate with an external debugger, for example over the switch. This is the same set of registers as the Debug Scratch registers in the processor status.

0x20 .. 0x27: Debug scratch

Bits	Perm	Init	Description
31:0	CRW		Value.

C.10 PC of logical core 0: 0x40

Value of the PC of logical core 0.

0x40: PC of logical core 0

Bits	Perm	Init	Description
31:0	RO		Value.

C.22 Chanend status: 0x80 .. 0x9F

These registers record the status of each channel-end on the tile.

0x80 .. 0x9F: Chanend status

Bits	Perm	Init	Description
31:26	RO	-	Reserved
25:24	RO		00 - ChannelEnd, 01 - ERROR, 10 - PSCTL, 11 - Idle.
23:16	RO		Based on SRC_TARGET_TYPE value, it represents channelEnd ID or Idle status.
15:6	RO	-	Reserved
5:4	RO		Two-bit network identifier
3	RO	-	Reserved
2	RO		1 when the current packet is considered junk and will be thrown away.
1	RO	0	Set to 1 if the switch is routing data into the link, and if a route exists from another link.
0	RO	0	Set to 1 if the link is routing data into the switch, and if a route is created to another link on the switch.

Bits	Perm	Init	Description
31:25	RO	-	Reserved
24	RW		Tristate processor mode pins.
23:18	RO	-	Reserved
17:16	RW		Processor mode pins.
15:4	RO	-	Reserved
3	RW	0	USB peripheral register access enable.
2	RW	0	USB interface block enable. Set to 1 to enable. Set to 0 to disable and reset all USB interface registers
1	WO	0	xCORE Tile reset. Set to 1 to initiate a reset of the xCORE Tile. This bit is self clearing. A write to this configuration register with this bit asserted results in no response packet being sent to the sender regardless of whether or not a response was requested.
0	WO	0	System reset. Set to 1 to initiate a reset whose scope includes most configuration and peripheral control registers. This bit is self clearing. A write to this configuration register with this bit asserted results in no response packet being sent to the sender regardless of whether or not a response was requested.

0x50: Reset and Mode Control

E.5 System clock frequency: 0x51

Bits	Perm	Init	Description
31:7	RO	-	Reserved
6:0	RW	25	Oscillator clock frequency in MHz rounded up to the nearest integer value. Only values between 5 and 100 MHz are valid - writes outside this range are ignored and will be NACKed. This field must be set on start up of the device and any time that the input oscillator clock frequency is changed. It must contain the system clock frequency in MHz rounded up to the nearest integer value. The following functions depend on the correct frequency settings: * Processor reset delay * The watchdog clock * The real-time clock when running in sleep mode * The USB clock (USB requires a 12, 24, 48, or 96 MHz oscillator)

0x51: System clock frequency

0x04: UIFM IFM control

Bits	Perm	Init	Description
31:8	RO	-	Reserved
7	RW	0	Set to 1 to enable XEVACKMODE mode.
6	RW	0	Set to 1 to enable SOFISTOKEN mode.
5	RW	0	Set to 1 to enable UIFM power signalling mode.
4	RW	0	Set to 1 to enable IF timing mode.
3	RO	-	Reserved
2	RW	0	Set to 1 to enable UIFM linestate decoder.
1	RW	0	Set to 1 to enable UIFM CHECKTOKENS mode.
0	RW	0	Set to 1 to enable UIFM DOTOKENS mode.

F.3 UIFM Device Address: 0x08

The device address whose packets should be received. 0 until enumeration, it should be set to the assigned value after enumeration.

0x08: UIFM Device Address

Bits	Perm	Init	Description
31:7	RO	-	Reserved
6:0	RW	0	The enumerated USB device address must be stored here. Only packets to this address are passed on.

F.4 UIFM functional control: 0x0C

0x0C: UIFM functional control

Bits	Perm	Init	Description
31:5	RO	-	Reserved
4:2	RW	1	Set to 0 to disable UIFM to UTMI+ OPMODE mode.
1	RW	1	Set to 1 to switch UIFM to UTMI+ TERMSELECT mode.
0	RW	1	Set to 1 to switch UIFM to UTMI+ XCVRSELECT mode.

F.5 UIFM on-the-go control: 0x10

This register is used to negotiate an on-the-go connection.

F.7 UIFM Serial Control: 0x18

0x18: UIFM Serial Control

Bits	Perm	Init	Description
31:7	RO	-	Reserved
6	RO	0	1 if UIFM is in UTMI+ RXRCV mode.
5	RO	0	1 if UIFM is in UTMI+ RXDM mode.
4	RO	0	1 if UIFM is in UTMI+ RXDP mode.
3	RW	0	Set to 1 to switch UIFM to UTMI+ TXSE0 mode.
2	RW	0	Set to 1 to switch UIFM to UTMI+ TXDATA mode.
1	RW	1	Set to 0 to switch UIFM to UTMI+ TXENABLE mode.
0	RW	0	Set to 1 to switch UIFM to UTMI+ FSLSSERIAL mode.

F.8 UIFM signal flags: 0x1C

Set of flags that monitor line and error states. These flags normally clear on the next packet, but they may be made sticky by using PER_UIFM_FLAGS_STICKY, in which they must be cleared explicitly.

0x1C: UIFM signal flags

Bits	Perm	Init	Description
31:7	RO	-	Reserved
6	RW	0	Set to 1 when the UIFM decodes a token successfully (e.g. it passes CRC5, PID check and has matching device address).
5	RW	0	Set to 1 when linestate indicates an SE0 symbol.
4	RW	0	Set to 1 when linestate indicates a K symbol.
3	RW	0	Set to 1 when linestate indicates a J symbol.
2	RW	0	Set to 1 if an incoming datapacket fails the CRC16 check.
1	RW	0	Set to the value of the UTMI_RXACTIVE input signal.
0	RW	0	Set to the value of the UTMI_RXERROR input signal

F.9 UIFM Sticky flags: 0x20

These bits define the sticky-ness of the bits in the UIFM IFM FLAGS register. A 1 means that bit will be sticky (hold its value until a 1 is written to that bitfield), or normal, in which case signal updates to the UIFM IFM FLAGS bits may be over-written by subsequent changes in those signals.

Bits	Perm	Init	Description
31:25	RO	-	Reserved
24	RO	1	Indicates that an ADC sample has been dropped. This bit is cleared on a read.
23:18	RO	-	Reserved
17:16	RW	1	Number of bits per ADC sample. The ADC values are always left aligned: 0: 8 bits samples - the least significant four bits of each sample are discarded. 1: 16 bits samples - the sample is padded with four zero bits in bits 3..0. The most significant byte is transmitted first. 2: reserved 3: 32 bits samples - the sample is padded with 20 zero bits in bits 19..0. The most significant byte is transmitted first, hence the word can be input with a single 32-bit IN instruction.
15:8	RW	1	Number of samples to be transmitted per packet. The value 0 indicates that the packet will not be terminated until interrupted by an ADC control register access.
7:2	RO	-	Reserved
1	RW	0	Set to 1 to switch the ADC to sample a 0.8V signal rather than the external voltage. This can be used to calibrate the ADC. When switching to and from calibration mode, one sample value should be discarded. If a sample value `x` is measured in calibration mode, then a scale factor `800000/x` can be used to translate subsequent measurements into microvolts (using integer arithmetic).
0	RW	0	Set to 1 to enable the ADC. Note that when enabled, the ADC control registers above are read-only. The ADC must be disabled whilst setting up the per-input-pin control. On enabling the ADC, six pulses must be generated to calibrate the ADC. These pulses will not generate packets on the selected channel-end. The seventh and further pulses will deliver samples to the selected channel-end. These six pulses have to be issued every time that this bit is changed from 0 to 1.

0x20: ADC General Control

H Deep sleep memory Configuration

This peripheral contains a 128 byte RAM that retains state whilst the main processor is put to sleep.

The *Deep sleep memory* is peripheral 3. The control registers are accessed using 8-bit reads and writes (use `write_periph_8(device, 3, ...)` and `read_periph_8 ↪ (device, 3, ...)` for reads and writes).

J Real time clock Configuration

The *Real time clock* is peripheral 5. The control registers are accessed using 32-bit reads and writes (use `write_periph_32(device, 5, ...)` and `read_periph_32(device, 5, ...)` for reads and writes).

Figure 52: Summary

Number	Perm	Description
0x00	RW	Real time counter least significant 32 bits
0x04	RW	Real time counter most significant 32 bits

J.1 Real time counter least significant 32 bits: 0x00

This registers contains the lower 32-bits of the real-time counter.

0x00: Real time counter least significant 32 bits

Bits	Perm	Init	Description
31:0	RO	0	Least significant 32 bits of real-time counter.

J.2 Real time counter most significant 32 bits: 0x04

This registers contains the upper 32-bits of the real-time counter.

0x04: Real time counter most significant 32 bits

Bits	Perm	Init	Description
31:0	RO	0	Most significant 32 bits of real-time counter.

K Power control block Configuration

The *Power control block* is peripheral 6. The control registers are accessed using 32-bit reads and writes (use `write_periph_32(device, 6, ...)` and `read_periph_32(device, 6, ...)` for reads and writes).

0x00: General control

Bits	Perm	Init	Description
31:10	RO	-	Reserved
9	RW	0	Set to 1 to switch USB suspend controller to USB power up enable.
8	RW	0	Set to 1 to switch USB suspend controller to power down enable.
7	RW	0	By default, when waking up, the voltage levels stored in the LEVEL CONTROL registers are used. Set to 1 to use the power-on voltage levels.
6	WO		Set to 1 to re-apply the current contents of the AWAKE state. Use this when the program has changed the contents of the AWAKE state register. Self clearing.
5	RW	0	Set to 1 to use a 64-bit timer.
4	RW	0	Set to 1 to wake-up on the timer.
3	RW	1	If waking on the WAKE pin is enabled (see above), then by default the device wakes up when the WAKE pin is pulled high. Set to 0 to wake-up when the WAKE pin is pulled low.
2	RW	0	Set to 1 to wake-up when the WAKE pin is at the right level.
1	RW	0	Set to 1 to initiate sleep sequence - self clearing. Only set this bit when in AWAKE state.
0	RW	0	Sleep clock select. Set to 1 to use the default clock rather than the internal 31.25 kHz oscillator. Note: this bit is only effective in the ASLEEP state.

K.2 Time to wake-up, least significant 32 bits: 0x04

This register stores the time to wake-up. The value is only used if wake-up from the real-time clock is enabled, and the device is asleep.

0x04: Time to wake-up, least significant 32 bits

Bits	Perm	Init	Description
31:0	RW	0	Least significant 32 bits of time to wake-up.

K.3 Time to wake-up, most significant 32 bits: 0x08

This register stores the time to wake-up. The value is only used if wake-up from the real-time clock is enabled, if 64-bit comparisons are enabled, and the device is asleep. In most cases, 32-bit comparisons suffice.

L Device Errata

This section describes minor operational differences from the data sheet and recommended workarounds. As device and documentation issues become known, this section will be updated the document revised.

To guarantee a logic low is seen on the pins DEBUG_N, MODE[4:0], TMS, TCK and TDI, the driving circuit should present an impedance of less than 100 Ω to ground. Usually this is not a problem for CMOS drivers driving single inputs. If one or more of these inputs are placed in parallel, however, additional logic buffers may be required to guarantee correct operation.

For static inputs tied high or low, the relevant input pin should be tied directly to GND or VDDIO.

M JTAG, xSCOPE and Debugging

If you intend to design a board that can be used with the XMOS toolchain and xTAG debugger, you will need an xSYS header on your board. Figure 54 shows a decision diagram which explains what type of xSYS connectivity you need. The three subsections below explain the options in detail.

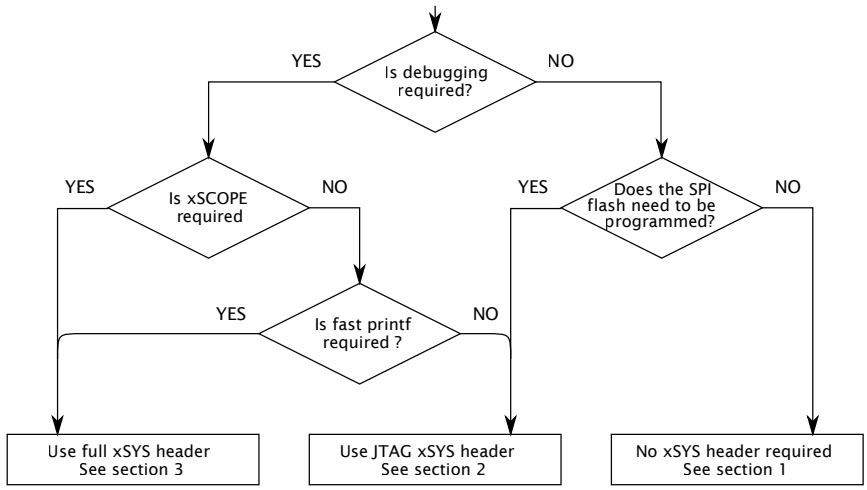

Figure 54: Decision diagram for the xSYS header

M.1 No xSYS header

The use of an xSYS header is optional, and may not be required for volume production designs. However, the XMOS toolchain expects the xSYS header; if you do not have an xSYS header then you must provide your own method for writing to flash/OTP and for debugging.