

GPIO – Summary of Major Features

NDK 5.7/MPTK 2.4

Application Note, 12 December 2007

Introduction

The PNX Family of processors includes a comprehensive set of General Purpose I/O (GPIO) pins that can be used for a wide variety of uses, from simple setting or reading of a single pin, to complex communications protocols that can be generated automatically. This Application Note attempts to summarize the major features of the GPIO pins and provides examples of some of the more common applications for this component.

This Application Note also summarizes some of the problems with the use of the advanced features of the GPIO pins. This Application Note shows where the advanced GPIO features cannot be used at all or cannot be used to implement certain protocols at high frequencies. TCS Help (Footprints) was used to identify where customers have the most questions and was used as a guide for providing examples in this document.

This Application Note is intended as a supplement to the PNXxxxx User Manuals. Please refer to the *Event Monitoring Mode* and the *Signal Monitoring & Pattern Generation Modes* in the "General Purpose Input Output Pins" or "GPIO" sections of the respective manual.

The strategy for implementing new designs is to use example programs as a reference.

1 GPIO Single-Pin Features

The GPIO pins can be read or written individually or in groups. In addition to direct access via MMIO registers, the NDK software provides the `tmhwGpio` and `tmdlGpio` components which provide different levels of abstraction of the underlying hardware. The `tmdlGpio` is the preferred interface to the GPIO pins, as it is thread-safe. The `tmhwGpio` interface is designed to be used in a single threaded environment before the OS is up.

Many of the GPIO pins have default uses, as described in the GPIO Pin List in the user manual. These pins are available for general use if their default function is not being used.

In addition, GPIO pins 11:8, 3:0 are used to determine the CPU boot mode. After the CPU has booted, these pins are then available for general use. This dual function is possible by using pull-up or pull-down resistors on the pins to enable their default function.

1.1 Manipulation using MMIO Registers

At the lowest level, the GPIO pins can be controlled by writing directly to the associated MMIO registers. This is sometimes necessary when the `tmhwGpio` or `tmdlGpio` components are not available, such as in bootscript or boot loader code (or for quick testing).

The mode for the GPIO pin must first be set via its respective GPIO Mode Control register. Access to the pin is then done using the GPIO Data Control register. Since each Data Control register controls 16 pins, a mask must be used to select which pin is being accessed. Note that more than one pin can be accessed in a single read or write operation using a suitable mask.

1.2 tmhwGpio Component

The `tmhwGpio` component provides the next level of abstraction. This component provides a useful interface for monitoring single pins, although the Event Queue interface can be accessed using this component. This component must be used in BSL code, where the operating system hasn't started yet, and thus `tmdlGpio` cannot be used.

The `tmhwGpio.h` file includes a description of the features and use of this component. The user manual has a chapter on peripheral IO using GPIO. An example program is also provided in `sd\od\osgeneric\comps\tmhwGpio\ex.`

1.3 tmdlGpio Component

The `tmdlGpio` component provides the highest level of abstraction for the GPIO pins. This is the preferred component to use in application programs since access to the GPIO pins is thread-safe.

The `tmdlGpio` component supports defining a group of pins that can be read or written simultaneously. This is done by defining a 'pins instance' (`tmdlGpioPinsGetInstanceSetup/tmdlGpioPinsInstanceSetup`), then using `tmdlGpioPinsRead` or `tmdlGpioPinsWrite`. These functions also provide an abstraction for reading from and writing to a single pin.

The example programs discussed below use the `tmdlGpio` component for pattern generation and signal sampling operations. The `tmdlGpio.h` file includes a description of the features and use of this component. The user manual has a chapter on peripheral IO using GPIO. An example program is also provided in `sd\od\osgeneric\comps\tmdlGpio\ex.`

2 GPIO Advanced Features Summary

In addition to the single pin read and write, the GPIO component includes four FIFOs (six in PNX85xx) for transferring bit streams. There is one DMA read channel and one DMA write channel, two DMA channels total, which are shared by the four FIFOs.

Each FIFO, referred to as an Event Queue, may be organized to output or input 1, 2, or 4 bits simultaneously. To implement continuous streams, dual memory buffers are used with each FIFO so that one buffer can be filled/emptied by the user program while the DMA controller is transferring data from/to the other buffer. Each memory buffer may be up to 1 MByte in size.

An interrupt is generated when the DMA transfer of the buffer is complete. This interrupt allows the user to update/empty the first buffer, while the contents of the second buffer are being transferred. This allows 'ping-ponging' of the buffers, preventing any interruption in the serial data stream. However, this also has the problem that the interrupt occurs only when a buffer is empty/full, and not when the last data have been transferred into/out of the shift register. See [Section 2.4, Transfer Complete Interrupts](#) below for more information.

The FIFO inputs/outputs may be assigned to arbitrary GPIO pins. Each 32-bit word in the FIFO is loaded into a shift register, which is then shifted to provide the serial stream. Bits must be interleaved by software if more than one bit is needed for the protocol.

2.1 Timestamping Mode

The GPIO component includes timestamping mode, which allows sending and receiving of a slow changing bit stream (single GPIO pin) by recording the time when the bit stream changes state. A sequence received using time stamping can be repeated exactly by outputting the same timestamp data. Each time stamp is a 32-bit word which represents a 1-bit direction (low-to-high transition or high-to-low) and a 31-bit time of when the transition occurred.

The timestamp clock frequency is 13.5 MHz, which gives better than 75 ns resolution for pin transitions. For receive, after DMA transfer from the FIFO to memory, software must reconstruct the bit stream pattern from the timestamp data. For example, this method is for receiving UART data.

2.2 FIFO Length

The length of each FIFO is fixed at 64 bytes (this is equal to the cache line size) and can only be read or written using one of the DMA controllers. Each memory buffer must be aligned to be a multiple of 64 bytes. Each FIFO requires two buffers.

Even though less than 64 bytes may be used in a transfer through the shift register, the DMA transfer is still 64 bytes. Because of a bug, the shortest buffer transfer length is two words. Bit-banging must be used to handle the odd word or bytes at the beginning or end of a transfer. For example, bit-banging would be used when transferring a small number of bytes to SD devices using the SPI protocol.

2.3 Endianness

Each 32-bit word is transferred from the FIFO to the shift register, then shifted out. The shift direction and endianness are fixed. The shift direction is MSB out, LSB in. This shift order determines the little endian order. The endian order used by the Event Queues is fixed and is not affected by the endianness setting of the CPU.

This little endianness order causes problems for byte oriented protocols. The bytes need to be shifted out in 0123 byte order, but because of the implicit endianness will be stored in the shift register in 3210 byte order. This means that the driver software must shuffle the bytes to put them in the correct order prior to shifting. This byte order correction must also be done for interfaces that control more than one bit.

For example, UART data must be sent in 0123 order, so the bytes must be shuffled to place the least significant byte in the most significant FIFO byte position, etc., before each word is stored in the memory buffer.

2.4 Transfer Complete Interrupts

The PNXxxxx will interrupt when a DMA buffer has been transferred to the FIFO, signaling when a buffer needs to be filled by the software. If it is important to know when the end-of-transfer (EOT) of a multi-block transfer is completed, then the software must implement a scheme to count the number of interrupts to determine when all of the buffers have been transferred.

The last interrupt occurs when the last block has been transferred to the FIFO. However, there is data still being transferred from the FIFO to/from the target after this last interrupt. Therefore, the software must wait for the time it takes to transfer the last bit to/from the FIFO.

As a variation, the PNX8550 provides an interrupt when the last bit of the last word of each block has been shifted out. However, the software still needs to count interrupts (i. e., count blocks) to know when the last block has been transferred.

2.5 Memory Latency Issues

Because the DMA controllers (one for read and one for write) are shared among the four FIFOs, the FIFOs must wait for each other if there is more than one active FIFO request at a time. This limits the maximum throughput and the memory latency for the transfers. Also, DMA requests from different sources must wait for the completion of transfers of the other DMA controllers in the system.

The "GPIO Frequency Restrictions" section in the manual discusses the maximum sustained frequency for time stamping and signal sampling (one FIFO is enabled), given the minimum guaranteed system latency of 40 microseconds.

For time sampling, the maximum sustained frequency:

$$16 \text{ samples} / 40 \text{ microseconds} = 400 \text{ kHz.}$$

For signal sampling, the maximum sustained frequency is:

$$512 \text{ samples} / 40 \text{ microseconds} = 12.8 \text{ MHz.}$$

Note that the sustained frequency is reduced when more than one FIFO is active. The minimum guaranteed system latency is dependent on DDR clock speed, the number of DMA requests pending in the system, and the performance of the memory device (wait states). The calculation of the minimum latency is beyond the scope of this document.

3 Problems Using Event Queues

The Event Queues were designed for use with remote control protocols. These protocols are tolerant of missing bits or words in the bit stream. However, other protocols, such as RS-232 and SPI, are intolerant of bit or word problems.

3.1 Pattern Generation Clocking

When starting the pattern generation clock, the startup is handled by a circuit in the clock generation block, outside of the GPIO hardware. This avoids the possibility of accidentally generating short pulses which may get the logic into an unwanted state. Unfortunately, this means that several clocks will be output before the data stream starts. Thus, it is not possible to guarantee that data output starts exactly when the clock starts. Also, the GPIO output pin will be held low until the data are transferred from the FIFO to the shift register (one clock delay).

To ensure that the protocol clock and output data are synchronized, the clock seen by the external circuitry must be output as part of the bit pattern. Then the GPIO clock is just used to transfer out the pattern and is not used in the protocol. See [Section 4.2, SPI](#) below for a discussion of this strategy.

3.2 Signal Sampling

3.2.1 Clock Synchronization Issue

Depending on the clock rate, from two to seven clocks may transpire before the first bit is transferred into the input shift register. This is caused by the cross-over clock circuit, which switches from internal to external clocks in such a way as to eliminate very short clocks. The number of delay clocks is based on the ratio of the internal and external clocks. The number of delay clocks can be determined empirically and all subsequent samples will have the same delay. However, exact synchronization of the input clock and data is not possible.

3.2.2 'Duplicate Last Word' at Input

Approximately 1% of the time, the sampling hardware will insert the last word of the previous sample in front of the first word of the new sample, causing a duplication of the prior 32-bit word. This means that the protocol must provide a means to detect the faulty reception and implement a retry strategy. This occasional word insertion means that signal sampling cannot be used for protocols such as RS-232 and byte-oriented SPI where retry is not possible.

Signal sampling can still be used for an SPI protocol if some form of error detection and retry is implemented. See [Section 4.2, SPI](#) below.

3.2.3 Work Around – Bit-Banging

In those cases where the GPIO signal sampling can't be used, the user can always toggle or read the GPIO pins under program (software) control. This is referred to as 'bit-banging', in that the program forces the GPIO pins to toggle rather than the hardware doing the toggling/reading automatically.

The effective clock frequency of bit-banging (determined by how fast the program can execute) will be much slower than using the automatic transfer of an Event Queue. Also, protocols using the same clock for input and output must use bit-banging for generating the output stream and reading the input stream.

3.2.4 Work Around – Timestamp Mode

Timestamp mode detects transitions on a single input pin, with a time resolution of about 75 nanoseconds. This means that timestamp mode can be used for protocols with relatively slow frequencies (below 400 kHz), such as RS-232. In this case, transitions are detected on the input data stream, without need for an external clock. Software must reconstruct the input stream by processing the timestamp data to detect signal transitions.

3.3 FIFO Length and Access Limitations

Fixing the FIFO length to 64 bytes and requiring access to the FIFO using DMA greatly restricts the flexibility of the Event Queues for short transfers. In some cases a protocol could have been implemented more efficiently if the FIFO length was variable and the FIFO could be accessed directly instead of only with DMA. This limitation also determines whether bit-banging would be more efficient for short transfers.

3.4 No Pattern Generation/Signal-sampling Clock Symmetry

Pattern generation uses an internal clock, which may also be configured to be output to a GPIO pin. Signal sampling uses an external clock, and so can use the pattern generation clock, allowing synchronization of signal sampling to pattern generation.

However, the opposite is not supported: an external signal sampling clock cannot be used to drive the pattern generation FIFO. This means that the Event Queues cannot be used to operate the Trimedia as a slave in some protocols such as SPI.

4 Examples of Using GPIO Pins

Simple examples for the use of the GPIO device library and hardware API are provided in the MPTK package. These examples are a reasonable starting point for most custom applications. The following sections provide more details for selected applications.

4.1 Remote Control

The `tmd1Rcc` component provides the interface for a remote control. The `exRcc` example program uses an Event Queue in timestamp mode to read a stream from a GPIO pin, then assembles and prints the received code. The GPIO pin definitions may need to be changed to match your target's hardware configuration. The various GPIO pins have different default characteristics, requiring

care to make sure the pins are set up correctly. This component provides a fully supported software driver for the receipt of remote control commands. Software for the inverse, an IR blaster, is not regularly supported nor available. It has been demonstrated in some products.

The following is from the PN15xx User Manual:

"PN15xx Series uses the GPIO pin event sequence timestamping mechanism and software to interpret remote control commands. The event sequence timestamping can resolve events on signal edges with 75ns accuracy. A sequence of events followed by an period of inactivity causes generation of an interrupt. Software then interprets the 'character' by looking at the event list consisting of (time, direction) encoded in memory.

"This allows interpretation of a wide variety of Remote Control protocols. The Philips RC-5, RC-6 and RC-MM remote control protocols are all decoded with this mechanism, provided that the RF modulation is performed externally. Most other Consumer Electronic vendor remote control protocols can be supported by appropriate software.

"Similarly, the event generation mechanism can be used to implement IR blaster capability. In this case, the modulator is included - the software generated pulses can be superimposed on an internally generated carrier."

4.2 SPI

The SPI protocol is a simple three-wire interface:

- SCK (shift clock),
- MISO, (master-in, slave-out),
- MOSI (master-out, slave-in).

Some applications use the chip select to gate the clock, and the assertion of chip select resets the SPI state machine. This means that the state machine starts over each time, so that any problem with one message is corrected for the next message. Without some form of reset or other means of synchronization, no false transitions or incorrect words can be tolerated for the protocol.

The SPI protocol defines a single clock that is routed to shift registers in both the master and the slave. On each (programmable) clock edge, the master shifts out a bit of data to the slave while simultaneously the slave shifts out a bit of data to the master. The slave is always dependent on the master clock for data transfers.

Because the master and the slave transfer data on the same clock edge, it makes sense to configure the Event Queue pattern generation clock to output to a GPIO pin and route this clock to the Event Queue signal sampling FIFO input clock pin. The first problem with this is that the pattern generation clock starts before the data starts, immediately causing loss of synchronization. To ensure the clock and data are in sync, the clock signal can be embedded in the FIFO data along with the output data, with the pattern generation FIFO configured to output to two GPIO pins. Rather than algorithmically interleave the clock with the data, a table can be used to pick one of 256 interleaved patterns for any 8-bit data value. Each 8-bit value requires a 32-bit word which includes the same data bit for two clock transitions, with alternating clock values (clock is coded in odd numbered bits):

1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
0	d7	1	d7	0	d6	1	d6	0	d5	1	d5	0	d4	1	d4	0	d3	1	d3	0	d2	1	d2	0	d1	1	d1	0	d0	1	d0

4.2.1 Trimedia as SPI Master

The Trimedia could use pattern generation to output data to the slave. However, because of the ‘duplicate last word’ problem in the GPIO signal-sampling hardware, signal sampling can only be used if the user's protocol can tolerate receive errors from the slave. For example, when receiving a stream of data, a checksum or CRC must be embedded in the stream. If the computed checksum or CRC does not match the received data, then a retry of the last message needs to be done. If retry is not possible, then signal sampling cannot be used for this application.

4.2.2 SPI Transfer Rate Using Bit-Banging

Note that with bit-banging, a test program has achieved a maximum transmit rate of about 2 MBit/sec and a receive rate of about 1.5 MBit/sec, with 100% CPU load. Higher transfer rates (or systems requiring lower CPU loading) require implementing the SPI protocol in external hardware.

There is currently no full-featured, general-purpose SPI driver supplied in the MPTK software.

4.2.3 tmdISd

The `tmdISd` component, which interfaces to an SD memory device, can be configured to use the SPI protocol for communication with the device. Here, bit-banging is used to for transferring commands to the slave and reading status from the slave.

Pattern generation is used to transfer data to the device. Pattern-generation mode outputs on two pins: the clock pin and the data out pin. Embedding the clock in the pattern ensures clock synchronization with the data. Each sector written to the device includes a CRC which is checked when the sector is read back. During a write operation, data shifted out from the slave can be ignored by the master.

For read operations, bit-banging is used to read less than a sector of data, while signal sampling is used for larger transfers. The pattern generation clock is used for the transfer, but in this case the data sent to the slave can be ignored by the slave. After receipt of a sector, the driver computes the CRC and compares it with the CRC stored in the sector. The software attempts to re-read the sector if the CRC is not correct. This works around the ‘duplicate last word’ problem in signal sampling.

4.2.4 Trimedia as SPI Slave

Because of the ‘duplicate last word’ problem in the GPIO signal-sampling hardware, signal sampling is not a reliable method for receiving data from an external master. A second issue (Clock Symmetry Problem) is that even though the signal sampling Event Queue can accept an external clock, there is no way for that external clock to be routed to the pattern generation Event Queue, which is required to return data to the master as data are clocked into the slave.

Bit-banging can be used when operating as a slave, but there needs to be a mechanism to know when the transfer begins. The input clock pin can be routed to a timestamp unit, which would generate an interrupt (`DATA_VALID`) to start the transfer. However, this has the disadvantage that the interrupt occurs for every clock signal, which limits the clock frequency to the interrupt latency of the system, in that the interrupt must be serviced before the next clock occurs.

For each clock transition (programmable for high-to-low or low-to-high), the software reads the GPIO input pin and outputs the next bit of data on the GPIO output pin. The input and output data must be programmatically shifted to convert the bit data to byte data.

4.3 UART

Signal sampling cannot be used for the UART driver because of the ‘duplicate last word’ in signal sampling. Instead, timestamp mode is used to detect transitions on the receive data input pin.

The `<Nexperia-root>\prod\mptk\comps\tmdlUartGpio` component uses pattern generation mode for transmit and timestamp mode for receive. This component has been tested to operate up to a baud rate of 460800.

References to Data Books

- [1] PNx15xx Series Data Book, Philips Semiconductors, Rev. 3 - 17 March 2006.
- [2] PNx17xx Series Data Book, Philips Semiconductors, Rev. 1 - 17 March 2006.
- [3] PNx8550 Programmable Source Decoder with Integrated Peripherals, Rev. 03 - 18 October 2005.