APPLICATION NOTE

A Discussion of the AS²Cwire™ Single-Wire Interface

Introduction
The Advanced Simple Serial Control single-wire (AS²Cwire) interface offers the simplest control possible for programmable power I/C devices. Using only a single wire, the AS²Cwire serial interface is designed to be driven by a wide range of systems in a simple, flexible way. This interface benefits system designers by bringing to market digitally programmable, high performance power I/Cs with minimal pinouts and tiny packaging.

AS²Cwire is easily integrated into a wide range of portable systems. The interface does not have a strict timing protocol, simplifying integration for software systems and eliminating any need for specialized hardware. The interface is simple, flexible, and compatible with processor chips found in nearly all of today’s portable systems.

This Application Note provides details about programming Skyworks power management I/Cs with the AS²Cwire single-wire interface are provided in the following discussion. Insight is also provided for system integration.

AS²Cwire Details
AS²Cwire is not defined by a strict timing and signaling scheme, but rather by a simple mechanism of toggling a line to program an AS²Cwire device. By avoiding strict timing requirements, the AS²Cwire interface remains flexible. The method for programming a device is simple; it consists of toggling the line of an I/O port so that rising edges are received by the EN/SET pin of an AS²Cwire device. This can be done with the digital I/O ports found in many processors, or any other digital or analog output capable of sending a line high and low in a pulsed fashion. Toggling the line submits edges to the AS²Cwire device, where each rising edge is counted.

Timing is flexible due to the interface’s generous tolerance level. The rising edges that program a part can be fast, slow, or inconsistently spaced. Timing of the edges is bound only by the loose timing specification. The AS²Cwire interface is simple to integrate with complex, multi-tasking systems, as well as slow, isolated subsystems dedicated to system management.

Voltage Levels
Two voltage level specifications are associated with the AS²Cwire serial interface. The high-impedance input at the EN/SET pin detects logic high and logic low levels when the high and low voltage level thresholds are reached for the specified times.

The high and low threshold levels are clearly defined in the product Data Sheets as a minimum and a maximum specification, respectively. Table 1 provides an example of the voltage level specifications found in most AS²Cwire device Data Sheets.

For VIH, the enable high threshold is given as 1.4 V. This means that the applied enable high signal must be 1.4 V or greater. For VIL, the enable low threshold is given as 0.4 V. This means that the enable low signal must be 0.4 V or lower. To submit a rising edge to the EN/SET pin, the line should be taken below 0.4 V and then taken above 1.4 V in a repetitive fashion. See Figure 1 for an illustration of this process.

Table 1. Example of EN/SET Threshold Specifications

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Conditions</th>
<th>Minimum</th>
<th>Typical</th>
<th>Maximum</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Enable threshold low</td>
<td>V_IN = 2.7 V</td>
<td>0.4</td>
<td></td>
<td></td>
<td>V</td>
</tr>
<tr>
<td>VIH</td>
<td>Enable threshold high</td>
<td>V_IN = 5.5 V</td>
<td>1.4</td>
<td></td>
<td></td>
<td>V</td>
</tr>
</tbody>
</table>
Table 2. Example of EN/SET Timing Specifications

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Conditions</th>
<th>Minimum</th>
<th>Typical</th>
<th>Maximum</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>t(EN/SET)_LO</td>
<td>EN/SET low time</td>
<td></td>
<td>0.3</td>
<td>75</td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>t(EN/SET)_HI_MIN</td>
<td>Minimum EN/SET high time</td>
<td></td>
<td>50</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t(EN/SET)_HI_MAX</td>
<td>Maximum EN/SET high time</td>
<td></td>
<td></td>
<td></td>
<td>75</td>
<td>µs</td>
</tr>
<tr>
<td>t_OFF</td>
<td>EN/SET off timeout</td>
<td></td>
<td></td>
<td>500</td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>t_LAT</td>
<td>EN/SET latch timeout</td>
<td></td>
<td></td>
<td>500</td>
<td></td>
<td>µs</td>
</tr>
</tbody>
</table>

Timing

A few timing specifications are associated with the AS2CWIRE™ interface. These relate to how fast or slow the interface can detect rising edges, as well as when to expect the data registers to update. Internally, a counter counts the rising edges and, after the line has been held high, the data is latched and the addressed register(s) are updated. At this point, the Digital-to-Analog Converter (DAC) and, therefore, the output are updated.

Table 2 provides an example of the EN/SET timing specifications found in most AS2CWIRE™ product Data Sheets. The specifications that correspond to edge detection are the t(EN/SET)_LO and t(EN/SET)_HI. To determine the t(EN/SET)_LO time, the Data Sheet gives a minimum and a maximum specification for t(EN/SET)_LO. This results in a straightforward set of limits. To ensure edge detection, the t(EN/SET)_LO time must be at least the minimum time and no greater than the maximum time. If edges are submitted outside of this range, there is a risk that the edges are not detected or that a latch timeout occurs when not intended (see Figure 2).

To determine the t(EN/SET)_HI time, the Data Sheet gives two specifications. There is a typical value for t(EN/SET)_HI_MIN and a maximum value for t(EN/SET)_HI_MAX. To determine the minimum t(EN/SET)_HI time, double the typical value for t(EN/SET)_HI_MIN. To determine the maximum value for the t(EN/SET)_HI time, use the maximum value from the t(EN/SET)_HI_MAX specification. If edges are submitted outside of this range, there is a risk that the edges are not detected or that a latch timeout occurs when not intended.

The Data Sheet gives two additional specifications that detail the AS2CWIRE™ interface. One is t_LAT, which is given as a maximum specification. This means that after submitting data and then holding the line high, the latch timeout is guaranteed to occur by this time. Typically, the t_LAT timeout occurs more towards the middle of the window created by t(EN/SET)_HI_MIN and maximum t_LAT (e.g., between 75 µs and 500 µs). The output updates after t_LAT, as illustrated in Figure 2.

The last relevant timing specification is t_OFF. The t_OFF time is given as a maximum specification. This means that after taking the line low, the part is guaranteed to enter shutdown within this time. Note that when the part is shut down, all registers are reset to zero as shown in Figure 3.
Programming

Programming an AS2Cwire device is simple. This section details what is involved and is intended to be general so that it applies to all AS2Cwire devices. A basic block diagram for a sample device, used to facilitate the programming discussion, is shown in Figure 4. The available operating modes for this device are presented in the same fashion as found in the Data Sheet for a real device.

The example device has two simple operating modes: it can operate as a constant voltage source, or it can operate as a constant current source. In either case, the output voltage or output current is set by programming the device using the AS2Cwire interface. There are 16 programmable settings available for each mode and each mode is assigned an address. The available settings are defined in Tables 3 and 4.

In the Data Sheet of an AS2Cwire device, the complete functionality of the device is detailed in Tables similar to those shown in Tables 3 and 4. In this case, the first Table lists the addresses for each mode register and the second Table lists the settings for each register. The example device has a voltage register and a current register. The voltage register is assigned Address 0 and the current register is assigned Address 1.

Programming an output voltage or current is accomplished by submitting address edges followed by data edges and then holding the EN/SET line high. The registers maintain their value while EN/SET remains high or until they are reprogrammed. Taking EN/SET low shuts down the device and resets all of the registers to zero. If no edges are clocked-in, but EN/SET is taken high, the device operates in the default state (Address 0, Data 1 or 5.0 V in this case).

Programming Examples

A few simple programming examples illustrate some of the details of operation. As a first example, assume that an event occurs and it is necessary to enable the constant current source output to 30 mA. The output is programmed to this level by submitting the appropriate number of address edges to the EN/SET pin, followed by the appropriate number of data edges.

Table 3 indicates that 18 edges are required to address the current register. From Table 4, the 30 mA setting corresponds to Data 16. This information indicates that the user must address the current register with 18 edges, and then program the current to 30 mA with 16 edges in that sequence.
Table 3. Register Addresses

<table>
<thead>
<tr>
<th>Address</th>
<th>EN/SET Edges</th>
<th>Addressed Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>17</td>
<td>Voltage</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>Current</td>
</tr>
</tbody>
</table>

Table 4. Mode Settings

<table>
<thead>
<tr>
<th>Data</th>
<th>Voltage (V)</th>
<th>Current (mA)</th>
<th>Data</th>
<th>Voltage (V)</th>
<th>Current (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.00</td>
<td>0</td>
<td>9</td>
<td>3.00</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>4.75</td>
<td>2</td>
<td>10</td>
<td>2.75</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>4.50</td>
<td>4</td>
<td>11</td>
<td>2.50</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>4.25</td>
<td>6</td>
<td>12</td>
<td>2.25</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>4.00</td>
<td>8</td>
<td>13</td>
<td>2.00</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>3.75</td>
<td>10</td>
<td>14</td>
<td>1.75</td>
<td>26</td>
</tr>
<tr>
<td>7</td>
<td>3.50</td>
<td>12</td>
<td>15</td>
<td>1.50</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>3.25</td>
<td>14</td>
<td>16</td>
<td>0</td>
<td>30</td>
</tr>
</tbody>
</table>

Figure 5. Turn-On Current Output

After clocking in 18 edges for Address 1, waiting for an amount of time equal to tLAT, and then clocking in 16 edges for data, the device turns on the output to 30 mA, as shown in Figure 5.

Next, assume that a different event occurs and it is necessary to turn on the voltage output to 4.25 V. Since Address 0 is the default, there are two ways to program the output to 4.25 V. The first is to address the voltage register and then clock in edges for the desired data setting; the alternative is to skip the address edges and just clock in data.

In Table 4, 4.25 V corresponds to Data 4. To bring the device out of shutdown and turn on the voltage output to 4.25 V, clock in 17 edges to address the voltage register (Address 0), wait for an amount of time equal to tLAT, and then clock in 4 edges for Data 4. After the latch timeout, the output updates to the programmed output level, as illustrated in Figure 6.

The alternative is to program 4.25 V on the voltage source output by clocking in only data; this is possible because the device resets to Address 0 during shutdown. In other words, all that is needed is to clock in 4 edges and hold EN/SET high. After the latch timeout, the device turns on to the same state as described in the previous paragraph for the address and data sequence (see Figure 7).
Next, assume that the device is operating with both the voltage and current outputs active. An event occurs and it is necessary to keep the voltage output active, but turn off the current output. This is accomplished by clocking in the appropriate number of edges (i.e., EN/SET does not need to be pulled low to reset registers first). Address the current register and program it to zero. The sequence is to submit 18 address edges, wait tLAT, and submit 1 data edge. The current output turns off, and the voltage output remains unchanged, as shown in Figure 8.

Finally, the device needs to be shut down. The device has been in an operating state because the EN/SET pin has been held high. Simply bring the EN/SET line low to enter shutdown mode. After taking the line low, there is a delay (tOFF) before shutdown mode is entered (see Figure 9). Note that shutdown resets the registers to zero.

All of the information explained in the previous examples is illustrated in the single diagram shown in Figure 10, a more general timing diagram used in product Data Sheets. The central theme is that to program an AS2Cwire device, the user submits a set of address edges, pauses for a period of time at least equal to tLAT, submits a set of data edges, and then holds the line high.

After tLAT has expired, the addressed data register is written, and the DAC is updated. To put the part into shutdown mode, simply take the EN/SET pin low. When tOFF has expired, the part enters shutdown and all of the registers are reset.

In summary, programming or reprogramming an AS2Cwire device simply amounts to addressing the desired register and then programming it to the desired setting. To ensure that AS2Cwire programming is successful, make certain that the signals fall within the timing specifications and that the signals achieve the voltage level thresholds. To turn the part off, the EN/SET pin is simply pulled low.
Figure 8. Turn-Off Voltage Output

Figure 9. Device Shutdown

Figure 10. Example of Data Sheet Timing Diagram
Software Integration

There are a number of practical ways to integrate AS²Cwire functionality into embedded software systems. Many of today’s cell phones and hand-held devices are built around low-power, high-performance 16- and 32-bit processors that run Real-Time Operating Systems (RTOSs) and application Operating Systems (OSs). These systems are balancing heavy loads with many hard, real-time deadlines. As a result, careful consideration is given to every additional piece of code added to a system. In line with today’s methodologies, AS²Cwire is flexible so that it is easy to integrate.

Due to the simplicity of the AS²Cwire interface, very little code is needed (see the example in Figure 11). Routines can just write 1s and 0s to an I/O port to toggle the line that is connected to the EN/SET pin of the part. The code can be fast and small, allowing it to be implemented as a high-priority task. A high priority task, dedicated to writing sets of edges, can run to completion extremely quickly since the AS²Cwire timing specification allows for fast timing. This fast execution time results in minimal impact to the software system.

Generally, there are two different programming sequences to deal with: there can be multiple bursts of edges that comprise address and data, or there can be a single burst of edges for only data (see Figure 12). For either case, the actual burst itself can be handled the same way by the software.

During the actual burst of edges, the code should execute with high priority. Between bursts, the execution priority is insignificant due to \( t_{LAT} \). There is no upper time limit for \( t_{LAT} \), but there is for \( t_{HI} \) and \( t_{LO} \). If using a preemptive, multi-tasking system, such as an RTOS, a high priority task can be used during the data burst. Then, the waiting between bursts (\( t_{LAT} \) between address and data) can occur at the application level. Since the AS²Cwire interface accepts fast timing, the high-priority task could toggle the I/O line quickly and minimize CPU time, as illustrated in Figure 13.

```
/* burst of edges runs with higher priority */
void onset_data(int data)
{
    signal_hipriority_task(TSK_AS2C, data);
}

/* program AATI part with address and data */
onset_data(address);  /* submit burst of address pulses */
wait(500);  /* wait 500us for T_LAT */
onset_data(data);  /* submit burst of data pulses */
wait(500);

/* program AATI part with data only */
onset_data(data);  /* submit burst of data pulses */
wait(500);  /* wait 500us for T_LAT */
```

Figure 11. Example of Code

```
for (i=0; i<data; i++) {
    0710 = 0;
    0910 = 1;
}
```

Figure 12. Single Burst and Multiple Bursts of Edges
Since the actual data burst is of primary concern, it is useful to estimate the execution time. The calculation focuses purely on the AS2Cwire pulse timing and does not account for any OS latencies. To minimize execution time, the fastest EN/SET timing should be used (300 ns for tLO and 100 ns for tHI). With this timing, the longest execution time occurs for the highest address.

In the preceding example, Address 1 is the highest address and requires 18 edges. This is the largest burst of edges and should be used to estimate the time, which can be calculated as follows:

\[ t_{RUN} = \max_{edges} \times (tL0(\text{MIN}) + tH0(\text{MIN})) \]

\[ t_{RUN} = 18 \times (300 \text{ ns} + 100 \text{ ns}) = 7.2 \mu\text{s} \]
From this calculation, it can be seen that the necessary execution time for the longest burst of edges is 7.2 μs.

A slow rate of edges can also be detected by the ASICwire interface. This gives a task dedicated to ASICwire a high degree of tolerance so that it can be preempted by more important code. However, a major concern is whether the task has a high enough priority. Using the previous example, the task should have a high enough priority that control returns within 75 μs (since this is the tEN/SET_HI_MAX and tLO maximum specifications). If the task is kept waiting for more than 75 μs, it could cause a delay within the burst of edges exceeding 75 μs. If tHi or tLO are allowed to exceed 75 μs, a tLAT or tOFF timeout may occur and interfere with the intended programming. A simplified example is shown in Figure 14.

To implement a delay between the bursts of edges (i.e., to delay tLAT between address and data), wait at the application level. There is no time limit for tLAT. For our example device, it is only necessary to wait at least 500 μs between address and data bursts; if the wait becomes much longer than 500 μs due to higher priority code, it is not a concern. Therefore, a sequence similar to the example shown in Figure 15 is sufficient.

An illustration summarizing ASICwire software implementation is shown in Figure 16. The ASICwire serial interface fits into commonly-used general multi-tasking RTOS environments, as well as custom-rolled software solutions tailored to a specific application. With little effort, ASICwire control can be easily integrated into today’s software/hardware platforms.

**Summary**

This Application Note has presented details of the ASICwire single-wire interface and possible approaches to software integration. The overall discussion is intended to offer additional insight towards using the interface in target applications and to clarify the conceptual aspects of the ASICwire single-wire interface. With this knowledge, designers can more effectively take advantage of the features and benefits offered by Skyworks digitally programmable, high performance power I/C devices.
Figure 16. AS²Cwire Implementation Summarized