Integração entre PSoC e Smartphone

Victor São Paulo Ruela
Universidade Federal de Minas Gerais

20 de Maio de 2015

Conteúdo

1 Introdução

O projeto consiste na integração de dados entre um system-on-ship (PSoC) e um aplicativo Android, utilizando para isso um Web Service na plataforma .NET. Através do aplicativo, o usuário consegue visualizar a temperatura do PSoC pressionando um botão. Também foi implementado um cliente Web para consumir o Web Service. Este cliente possui um botão para requisitar a leitura da temperatura do PSoC e a partir dessas leituras, constrói uma tabela contendo todos os valores lidos. Portanto, a seguinte arquitetura de integração foi implementada:

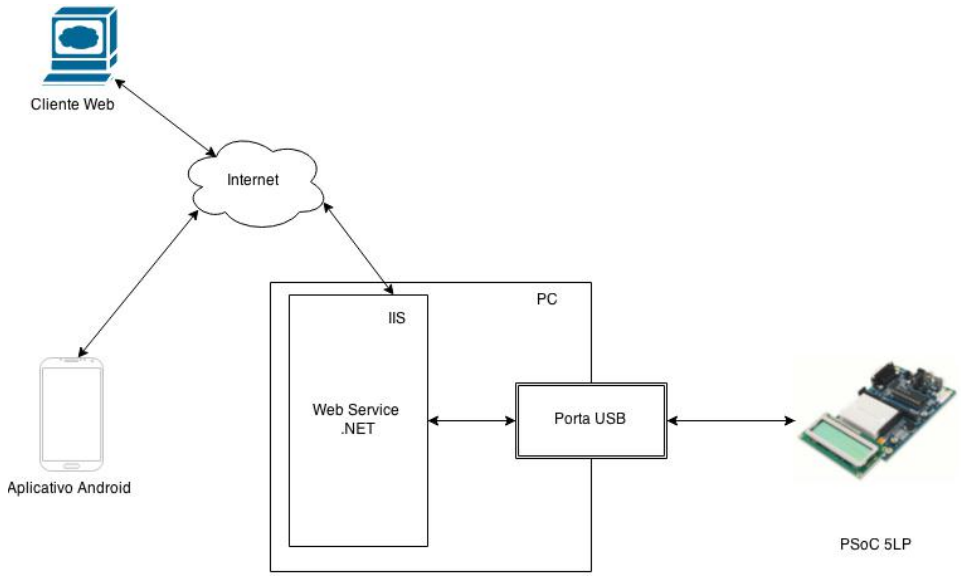

Figura 1: Arquitetura da integração

2 Implementação

2.1 Aplicação do PSoC

A aplicação foi feita no PSoC Creator. A criação de uma aplicação para o PSoC é feita através de diagrama de blocos e em C. Inicialmente, colocamos as instâncias dos componentes que serão utilizados em uma página do arquivo **TopDesig.cysch**, que é o padrão de qualquer projeto no PSoC Creator. Depois precisamos inicializar e interligar os diversos componentes através de suas APIs, em linguagem C. Para este projeto, foi foram necessários os componentes **DieTemp** e **USBUART**. O primeiro permite a leitura da temperatura do PSoC enquanto o segundo habilita o uso da porta USB da placa. Ambos foram usados com a configuração padrão.

O código em C utiliza as funções da API de cada componente. A lógica de funcionamento da aplicação é bem simples:

1. Inicializar o componente **USBUART**
2. Aguardar enumeração da porta USB
3. Aguardar mensagem de requisiçao de leitura
4. Obter temperatura através do bloco **DieTemp**
5. Enviar para a porta USB

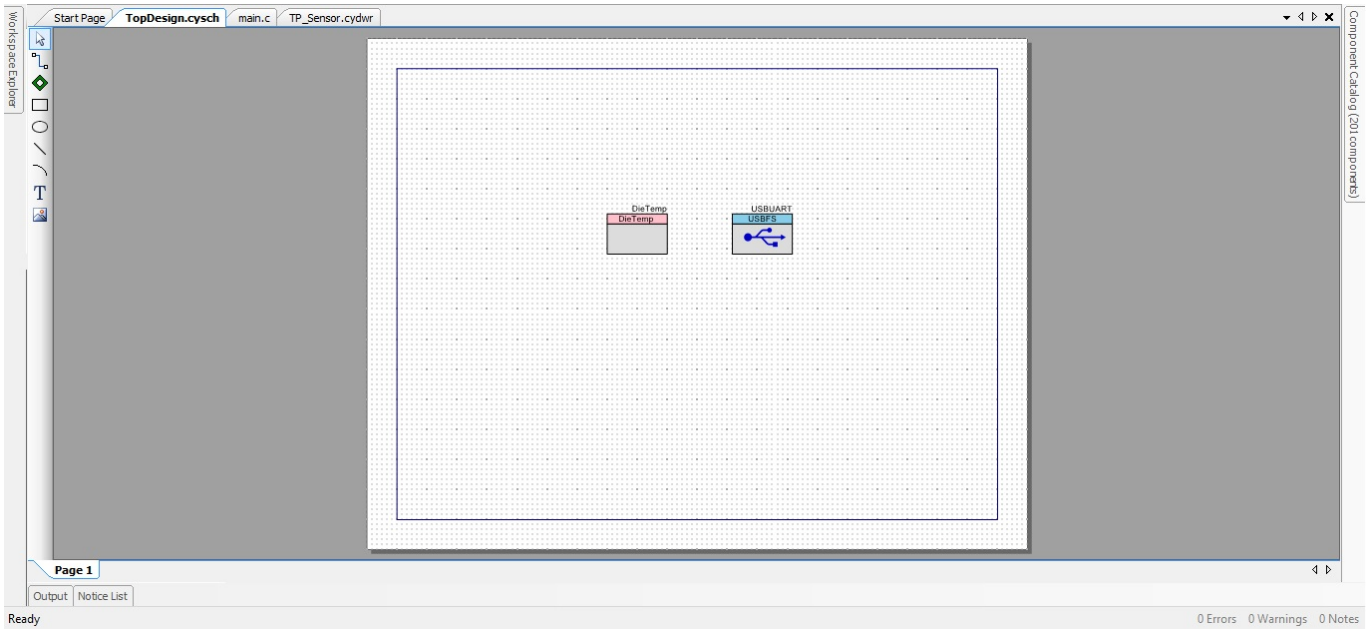

Figura 2: Diagram de blocos no PSoC

O PSoC fica em um loop infinito aguardando novas mensagens de requisição de leitura. O bloco USBUART funciona como uma porta serial COM, facilitando a leitura e escrita de dados pela aplicação **.NET** do Web Service.

2.2 Web Service

O Web Service está implementado na platforma .NET versão 4.0, usando a linguagem C#. Ele será publicado em um servidor web local através do **IIS - Internet Information Services**. Foi necessário realizar uma configuração do roteador: redirecionamento da porta 80 ao endereço de IP local da máquina conectada ao PSoC, para tornar o Web Service visível a todos os computadores e dispositivos conectados à rede. A publicação de Web Service foi facilitada com o uso do **MS Visual Studio 2012**, o qual possui uma ferramenta muito útil para a publicação de aplicações Web. Isso reduziu bastante o tempo necessário para desenvolver a aplicação, uma vez que não foram encontrados muitos problemas relacionados à configuração do IIS. O Web Service foi configurado para funcionar com o protocolo SOAP e os métodos GET e POST do HTTP.

2.2.1 Estrutura da aplicação

Decidi por implementar o Web Service usando ASMX, porque ele é mais simples e fácil de configurar, e atendia a todos os requisitos do projeto: uso do HTTP(SOAP) e do IIS. O WCF também poderia ser usado, mas é uma ferramenta mais avançada e poderosa. Como a aplicação desenvolvida é simples, seu uso se mostrou desnecessário. O Web Service fornece dois serviços:

ReadPSoC Serviço responsável pela comunicação com o PSoC através da porta USB, consumido principalmente pelo aplicativo Android. Inicialmente, ele obtém uma lista de valores de temperatura através da variável Session, que contém os dados de sessão para um determinado usuário. Feito isso, começa a troca de dados propriamente dita:

1. Criação do objeto SerialPort:

```
SerialPort s = new SerialPort("COM3", 9600);
```

O parâmetro COM3 é o número da porta COM e 9600 é a baudrate da porta, definida na configuração do bloco USBUART.

2. Abertura da porta serial:

```
s.Open();
```

3. Escrita e leitura na porta serial:

```
s.WriteLine("OK");
line = s.ReadLine();
```

O dado lido também é adicionado à lista descrita acima, para ficar disponível ao usuário da aplicação Web.

4. Fechamento da porta serial:

```
s.Close();
```

As funções acima fazem parte da biblioteca **System.IO.Ports** do .NET. O código possui tratamento de exceções para evitar que a leitura falhe e comprometa a integração do sistema. A principal exceção gerada ocorre quando tentamos ler com o PSoC desconectado do computador. Nesse caso, o Web Service envia a mensagem ”Error opening port”, indicando a falha. O serviço retorna uma string, contendo a temperatura lida e uma *time stamp* indicando o momento em que a leitura foi feita.

getReadings Serviço usado pela aplicação Web, que será explicada com mais detalhes à frente. Ela retorna uma lista de strings contendo as temperaturas lidas salvas na variável Session do usuário.

Ambos os servicos são definidos como **[WebMethod(EnableSession=true)]**. Esse parâmetro permite salvar e recuperar valores do usuário que está acessando página Web.

2.3 Aplicativo Android

O aplicativo foi implementado em Java, usando as bibliotecas do pacote Android SDK tools. A comunicação com o Web Service é feita com o protocolo SOAP.

2.3.1 Interface gráfica

A interface gráfica é bem simples, e consiste de um **EditText** [5] para o usuário digitar o IP, um **TextView** [6] para exibir a temperatura e um **Button** [4] para requisitar um serviço do Web Service. A interface gráfica será exibida na seção dos resultados nas figuras 5 e 4.

2.3.2 Estrutura da aplicação

O aplicativo foi implementado usando duas classes: **MainActivity** e **XMLResponseHandler**.

MainActivity Consiste na atividade principal (thread primária). É responsável por atualizar o IP digitado e realizar as ações necessárias quando o botão é pressionado. Quando isso ocorre, o seguinte código é executado:

```
public void onClick(View v) {

        URL = "http://" + editText.getText() +
           "/WS_TP/Service1.asmx/ReadPSoC";
        new HttpGetTask().execute();

}
```

Esse método cria uma thread secundária que inicia a comunicação com o Web Service. Esta é considerada uma boa prática em aplicativos Android, pois evita que a interface gráfica fique congelada enquanto a comunicação está sendo processada. A thread executa o seguinte código depois de criada:

```
private class HttpGetTask extends AsyncTask<Void, Void, String> {

        AndroidHttpClient mClient = AndroidHttpClient.newInstance("");

        @Override
        protected String doInBackground(Void... params) {
                HttpGet request = new HttpGet(URL);
                XMLResponseHandler responseHandler = new
                   XMLResponseHandler();
                try {
                        return mClient.execute(request,
                           responseHandler);
                } catch (ClientProtocolException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        return null;
        }
```

A thread é uma extensão da classe **AsyncTask** [3] e consegue comunicar-se com o Web Service criando, inicialmente, uma instância da classe **AndroidHttpClient** [2], que é uma implementação de um cliente HTTP configurado para o Android. Feito isso, são criadas uma instância da classe **HttpGet**, que implementa o método HTTPGET definido no protocolo HTTP, e da **XMLResponseHandler** que é responsável por processar a mensagem XML recebida. Finalmente, executamos o cliente HTTP passando os objetos anteriores como parâmetros.

XMLResponseHandler Classe utilizada para processar a mensagem recebida em XML. Ela utiliza a API **XMLPULL V1** para processar a XML e obter a informação desejada, no caso a temperatura do PSoC. Seu funcionamento é simples e direto, e está bem descrito em: `http://www.xmlpull.org/`. A informação extraída é salva na variável associada ao **TextView** que então atualiza a temperatura exibida na UI.

2.4 Cliente Web

O cliente Web foi implementado usando Web Forms ASP.NET. A página foi desenvolvida em HTML e contém uma breve descrição de sua funcionalidade, um botão para requisitar a leitura do PSoC e uma tabela mostrando os valores recentemente lidos pelo usuário. Ao clicar no botão, o seguinte script em C# é executado:

```
protected void btnRead_Click(object sender, EventArgs e)
{
        // Criar instancia do objeto ReadPSoCService
        ReadPSoCService.Service1SoapClient client =
                        new ReadPSoCService.Service1SoapClient();

        // Realiza leitura do PSoC
        string result = client.ReadPSoC();
        // Exibe o resultado na pagina
        lblResult.Text = result;

        // Associa a variavel com os valores da tabela a lista
           retornada pelo servico
        gvReadings.DataSource = client.getReadings();
        gvReadings.DataBind();

        // Altera titulo da tabela
        gvReadings.HeaderRow.Cells[0].Text = "Recent Temperature
           Readings";
}
```

Inicialmente, precisamos adicionar ao projeto no MS Visual Studio uma referência ao serviço fornecido pelo Web Service. Feito isso, criamos uma instância de um objeto representando o serviço acima. Logo, podemos acessar ambos os serviços definidos anteriormente (**ReadPSoC** e **getReadings**). **ReadPOsC** é usado para fazer a leitura do PSoC quando clica-se no botão e **getReadings** é necessária para montar a tabela com os valores lidos, pois ela acessa a variável contendo os dados de sessão do ususário.

3 Resultados

Todos os testes foram realizados com os dispositivos conectados à rede local.

3.1 Cliente Web

O cliente Web foi testado em diversos computadores e seu funcionamento foi como o esperado em todos os casos. A página web foi acessada de um computador com sistema operacional Windows, outro com Mac OS X e de smartphones, funcionando corretamente em todos.

3.2 Aplicativo Android

O aplicativo foi testado no emulador de dispositivos Android e em um Nexus 4. Ambos funcionaram perfeitamente.

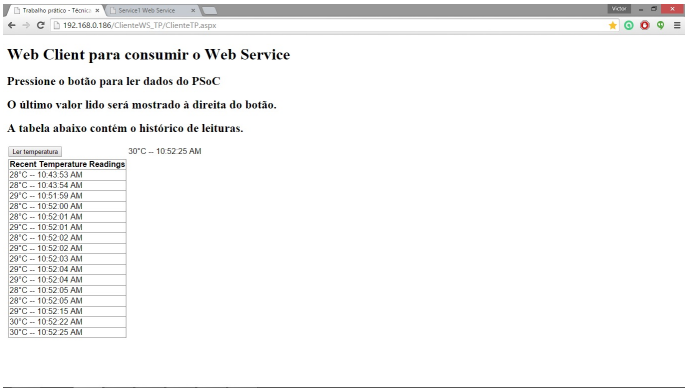

Figura 3: Funcionamento do cliente Web

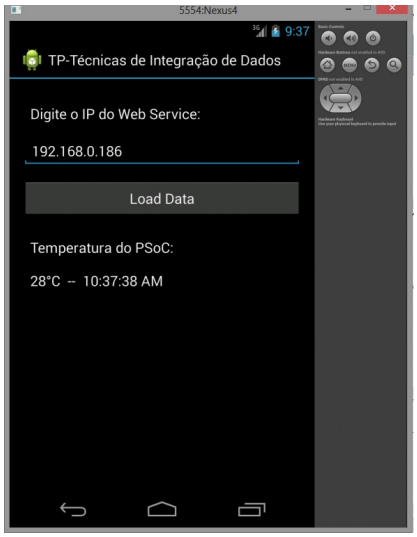

Figura 4: Screenshot do emulador de Dispositivos Android

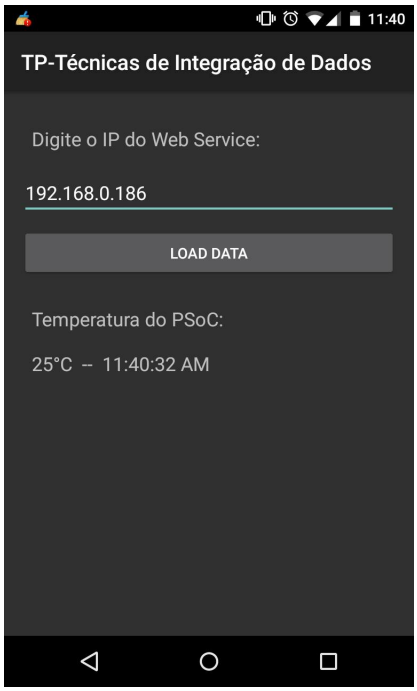

Figura 5: Screenshot de um Nexus 4

4 Dificuldades encontradas

A implementação do trabalho ocorreu sem muitos problemas, mas algumas dificuldades foram encontradas e serão descritas a seguir.

Driver USB do PSoC Este sem dúvidas foi o maior problema encontrado. Ao compilarmos e gravarmos o programa no PSoC, a porta USB para a troca de dados não é automaticamente reconhecida pelo Windows. Portanto, o driver deve ser instalado manualmente, porém ele não é disponibilizado de maneira fácil pela Cypress. Felizmente, no caso do bloco **USBUART**, o driver é gerado automaticamente na compilação do projeto e fica localizado na pasta "Generated_Source"do projeto, com o nome **USBUART_cdc.inf**. Logo, para resolver este problema é preciso ir no gerenciador de dispositivos do Windows e instalá-lo com o driver citado anteriormente.

Instalação do aplicativo no Smartphone A instalação do aplicativo no Nexus 4 utilizado nos testes também teve problemas relacionados ao driver. A forma inicialmente utilizada consistia em fazer o upload do aplicativo através do eclipse, com o celular conectado à porta USB. Isso permite debugar o aplicativo rodando no celular. Para isto, devemos baixar e instalar uma aplicação chamada ADB (Android Debug Bridge), a qual contém os drivers necessários. Entretanto, tive diversos problemas com o driver, o qual funcionou algumas vezes mas de uma hora para a outra parou de funcionar, me impossibilitando de carregar o aplicativo no Nexus 4. A solução para este problema foi fazer o upload do arquivo com extensão **.apk**, que é o aplicativo compilado, para o Dropbox, baixá-lo do Smartphone e por fim instalá-lo manualmente.

Leitura da porta USB Para a leitura/escrita dos dados da porta USB, foi utilizada a biblioteca **System.IO.Ports** da plataforma .NET. Seu uso é bastante simples, porém ela possui um comportamento que pode ser prejudicial a qualquer aplicação: ao requisitarmos o fechamento da porta, esta demora um intervalo indefinido de tempo para poder ser aberta novamente. Isso faz com que a porta fique bloqueada quando tentamos abrí-la novamente antes de ela ser efetivamente fechada. Esse problema foi observado durante os testes quando requisições eram feitas com um intervalo de tempo muito pequeno entre elas. Se isso ocorre, é preciso reiniciar o computador para conseguir novamente abrir a porta USB. Minha sugestão para evitar esse problema é adicionar uma pequena pausa ao programa, para que haja tempo suficiente para a porta ser efetivamente fechada.

Configuração do Web Service no IIS A publicação manual do Web Service no IIS não é tão simples e direta. Antes de descobrir a função **Publish** do Visual Studio 2012, tentei adicionar manualmente o Web Service e ocorriam diversos erros de configuração. Um deles estava relacionado às permissões de acesso à pasta que continha o projeto: é preciso adicionar permissão para que os grupos de usuários **IUSR** e **IIS_IUSRS** possam acessar o conteúdo da pasta. Isso é um requerimento do IIS. Minha sugestão é sempre usar a ferramenta **Publish**, pois ela já realiza todas essas configurações e reduz bastante o trabalho de realizar esta tarefa.

5 Conclusão

Através dos resultados exibidos anteriormente, podemos concluir que a integração entre o PSoC e aplicativo Android/Cliente Web foi realizada com sucesso. Algumas mudanças foram feitas em relação ao projeto original descrito no estudo de viabilidade:

- Utilização do MS Visual Studio 2012 para implementar o Web Service;
- O visor LCD não funcionou e foi retirado do projeto;
- Desenvolvimento de um cliente Web;
- Melhora da interface gráfica do aplicativo.

As mudanças realizadas aumentaram um pouco a complexidade do projeto, porém foram implementadas sem grandes problemas e acrescentaram bastante ao resultado final. Algumas melhorias poderiam ser feitas, e entre elas podemos citar a implementação do Web Service em **WCF** ao invés de **ASMX**, de forma a aumentar a robustez da aplicação.

O projeto implementado exemplifica uma forte tendência atual, que é a integração entre Smartphones e diversos tipos de sistemas através de Web Services. A idéia pode ser extendida para aplicações mais complexas e interessantes. Por exemplo, uma aplicação é a monitoração de sinais vitais de pacientes em tempo real, através de um aplicativo consumindo um Web Service. Isso permitiria aos médicos monitorar diversos pacientes de forma mais eficiente e remotamente.

Referências

[1] http://csharp-video-tutorials.blogspot.com.br/2013/11/part-1-introduction-to-aspnet-web.html, acessado em 13/05/2015.

[2] http://developer.android.com/reference/android/net/http/AndroidHttpClient.html, acessado em 13/05/2015.

[3] http://developer.android.com/reference/android/os/AsyncTask.html, acessado em 13/05/2015.

[4] http://developer.android.com/reference/android/widget/Button.html, acessado em 13/05/2015.

[5] http://developer.android.com/reference/android/widget/EditText.html, acessado em 13/05/2015.

[6] http://developer.android.com/reference/android/widget/TextView.html, acessado em 13/05/2015.

[7] Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley Professional, first edition.